Interface

2004
6
Contents



［表紙デザイン：㈱プランニング・ロケッツ］





特集

最新メカトロ技術と組み込み制御技術の結晶

ようこそ二足歩行ロボット制御の世界へ

Cover Story Welcome to the world of humanoid robots 43

Interface

2004 6

Contents

連載「開発技術者のためのアセンブラ入門」と「フリーソフトウェア徹底活用講座」は，お休みさせていただきます．

流通情報システムの総合展示会

SECURITY SHOW 2004/ IC CARD WORLD 2004

北村 俊之

「街作り・流通 ルネサンス」をテーマに，次世代の店舗・商業施設の街作りを提案する流通業向けの総合展示会が3月2日(火)～5日(金)の4日間，東京ビッグサイトで開催された．主催は日本経済新聞社．同展示会は，店舗の内装やディスプレイを提案する「JAPAN SHOP 2004」をはじめ，六つの展示会で構成されており，中でも流通情報システムをテーマにした「RETAIL TECH JAPAN 2004」，店舗やオフィスの防犯，安全管理を紹介する「第12回 セキュリティ・安全管理総合展 SECURITY SHOW 2004」，ICカードとICタグ，カード・ビジネスに関する展示会「IC CARD WORLD 2004」の三つがITビジネスに関連する展示会となっていた．この3展示会の合計出展社数は，のべ488社，出展小間数1491と，昨年を大きく上回る規模での開催となった．最終的な来場者数は，4日間で179,232人(3展示会合計)だった．今回は，「SECURITY SHOW 2004」と「IC CARD WORLD 2004」を中心にレポートする．

●「SECURITY SHOW 2004」

近年の「セキュリティ」ということばには，二つの意味合いがある．一つは，サーバやパソコン，企業情報などを保護する意味でのセキュリティ．もう一つは，ビルや集合住宅，個人住宅などを犯罪から保護する，防犯という意味合いのもの．「SECURITY SHOW 2004」は，後者に属する展示会である．近年では，セキュリティ機器のディジタル化により，センサや監視カメラなどの機器をまとめて制御できるシステムも多くなっている．またここ数年，この種のセキュリティの傾向として，指紋や顔，静脈，虹彩など身体の一部によって個人認証を行うバイオメトリクス認証技術を駆使したシステムも増えている．こうした状況を背景に，セキュリティ機器やサービスへの関心は非常に高くなっている．

東芝は，顔照合技術を入退室の認証に利用したセキュリティ・システム「FacePass」(写真1)の展示を行っていた．これは，あらかじめ登録してある顔の映像と入室時の顔の映像を比較し，本人かどうかを判定するというものである．このシステムでは，目や鼻の位置関係を記録して認証する方式をとっている．そのため，双子や顔立ちの似た兄弟でも見分けられるとのことである．また，顔データの登録内容を認証のたびに更新し，顔の経年変化を学習する機能も備えているという．静止画の認証では，顔の向きが異なると認証が難しい，写真によるなりすましが可能などの問題があったが「FacePass」では，静止画よりも情報量の多い動画を使い，判定精度を高めているということだ．



写真1　東芝のFacePass

日立エンジニアリングは，識別番号を入力して筒状の装置に指を差し込むことで，個人認証を行う指静脈入退管理システム「SecureVein Attestor」(写真2)を展示していた．指の血管パターンの画像をあらかじめ登録しておき，入室時に一致すればドアを開錠するというもの．簡単な操作で個人認証が可能であるため，オフィスやデータ・センタの入退室管理で多く利用されているという．このシステムは，指の爪側から当てた近赤外線で指の腹面側から血管パターンを読み取っているという．血液中のヘモグロビンは近赤外線を通さないという性質に着目し，指の表面近くの静脈を透かして読み取るしくみになっている．指静脈は指紋同様，他人と一致する確率が非常に低く，体の外側から見えない血管パターンで照合するため偽造が難しいとのこと．



写真2　日立エンジニアリングのSecure VeinAttestor

アイリス(虹彩)の模様で個人認証するゲート管理システム「アイリスパス-WG」(写真3)を展示していたのは，沖電気工業である．あらかじめ登録してある両目のアイリス模様が，入室時に一致すればドアが開錠するというしくみである．ほかのバイオメトリクス認証法と比較して，誤認識する確率が極めて低い点が特徴となっている．このシステムでは，瞳孔の周囲にあるドーナツ状の部分，アイリスの模様が個人ごとに異なることを認証に用いている．この模様を同心円状に分解してデータ化しており，部屋の明暗で瞳孔が拡縮したり，まぶたやまつげがあっても，高い精度で判別できるという．また，アイリスの模様は2歳を過ぎると加齢による経年変化が少なく，一度登録すればほぼ一生利用できるとのことである．



写真3　沖電気工業のアイリスパス-WG

● IC CARD WORLD 2004

交通定期券や電子マネーなどの分野に続いて，2003年8月には住民基本台帳カードの配布など，公共分野においてもICカードの活用が本格化してくる．その応用分野は，金融，セキュリティ，流通，医療など着実に拡大している．また，非接触ICカードなどのRFID技術を用いたICタグも，書籍やトレーサビリティ分野への導入に加え，アパレル分野など物流，SCM分野での活用が始まりつつあり，注目を集めている．今回，この分野で来場者の関心を集めていたのが，小型で低価格なICカード発行プリンタだった．各社，¥1,000,000前後の価格帯で提供しており，¥500,000を切るカード・プリンタを展示しているメーカもあった．学生証や社員証発行などの用途で，再発行が迅速に行えるなどのメリットで注目度が高いという．

ニスカは，低価格で省スペース設計の新製品「PR5x」シリーズ(写真4)を参考出品していた．同製品は，マイフェア，Felica，TN2，hitagなどさまざまな非接触ICカードに対応しており，磁気と非接触ICを同時にエンコードできるなどの特徴をもっている．また，磁気エンコーダは自走型を採用しているため，印刷との並列処理による連続発行処理時間の短縮，後から追加が可能など，用途に応じた使い分けに対応している．カードの自動反転機構や表裏両面印刷をサポートすることで，ランニング・コストを大幅に削減できることも大きな特徴の一つとしている．



写真4　ニスカのPR5xシリーズ

アイアンドディは，「DCP5300」，「DCP35」などの各種ダイレクト・カード・プリンタと顔写真入りIDカード発行管理システム「ID Maker」を組み合わせた，ICカード発行ソリューションを展示していた(写真5)．カード表面の個人情報は，社内のデータベース・サーバと連携できるなど，カード発行後の管理運用面でも，包括的なソリューションの提供が可能であるとのことだった．日本データカードは，小型で低価格なカード専用プリンタ「SP35」，「SP55」などのデモを行っており(写真6)，来場者の関心を集めていた．スマート・カードにも対応しており，ブースではFelicaカード発行のデモンストレーションも行われていた．



写真5　アイアンドディのダイレクトプリンタDCP5300/M/ICを用いたソリューション



写真6　日本データカードのカード専用プリンタ

Show & News Digest

トロンプロジェクトによるフォント・トレーサビリティ・システム

■日時：2004年3月10日(水)
■場所：メルパルクTOKYO(東京都港区)

トロンプロジェクトにより，印刷業界における外字問題を解決するためのフォント・トレーサビリティ・システムが発表された．

従来の文字コードに存在しない漢字を印刷業界などで使いたいとき，これまでは外字機能を用いて各社が個別に対応していたが，このとき割り振られる外字コードに関しては各社バラバラであり，統一的に管理されていないという問題があった．これを，外字に対してTRONコードを割り当て，統一的に管理できるようにしたものだ．

この外字コードをまとめたものを「外字表」と呼び，この外字表を管理するためにユビキタスIDセンターから付与されたユビキタスID(ucode)を用い，統一的に管理する．

これらの機能を使用することにより，A社の外字表を使ったデータをB社の外字表を使ったデータに変換することが容易になる．外字に関しては，TRON文字収録センターに申請すれば，一文字単位で追加できるとのことだ．

技術的背景としては，TRONコードがほか文字コード(SJISやUnicodeなど)を内部に包含することができる「メタコード体系」であることがあげられる．そのため，文字コードの拡張が容易であり，今回のような中間文字コードとしては適しているといえる．



トロンプロジェクト・リーダー
坂村 健氏

イーエルティ，GPLセミナーを開催

■日時：2004年3月25日(木)～26日(金)
■場所：トスラブ赤坂会議室(東京都千代田区)
■URL：http://www.emblit.co.jp/

(株)イーエルティは，GPLを正しく理解し運用することを目的としたGPLセミナーを開催した．

Linuxをはじめとするフリーソフトウェアの多くはGPL/LGPLで配布されることが多い．企業がこれらのソフトウェアを製品に組み込んで使用する場合には，ライセンスを正しく理解することが求められる．そこでこのセミナーでは，GPL/LGPLの条文を一行ずつ読み解き，これに日本における法解釈や判例などを紹介する，コメンタリ形式で解説が行われた．その際，著作権法だけでなく，特許，知的財産権，消費者保護などの幅広い観点から解説が行われるという，実用的な内容であった．

また，実際の運用面において，作成したコードをGPLにする場合やしたくない場合にどのような手段を用いれば良いのかという技術的な内容も紹介された．

同セミナーは定期的に開催され，次回は5月頃を予定している．


セミナーのようす

アイピーフレックスと富士通，ダイナミック・リコンフィギュラブル・プロセッサ「DAP/DNA-2」の受注を開始

■日時：2004年3月17日(水)
■場所：アイピーフレックス本社(東京都品川区)

アイピーフレックス(株)と富士通(株)は，システムが動作中でも内部の回路構成を1クロックで切り替えられるダイナミック・リコンフィギュラブル・プロセッサ「DAP/DNA-2」の受注を開始した．5月中旬よりサンプル品の出荷を開始する．価格は個別見積もり．また，統合開発環境として「DAP/DNA-FW II」も用意している．こちらの価格は350万円．

本プロセッサは，回路構成を切り替える制御を行う32ビットRISCコア「DAP(Digital Application Processor)」と，166MHzで動作する376個の専用演算器(PE：Processing Element)部分「DNA(Distributed Network Architecture)」で構成されている．専用演算器の構成を切り替えることにより，従来はアプリケーションごとに用意していた複数のチップの機能を1チップで実現できる．



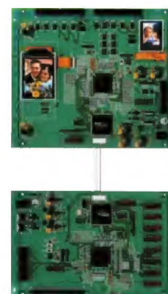
DAP/DNA-2

エプソンとルネサスが策定した携帯電話向けシリアル・インターフェース仕様が無償で公開に

■日時：2004年3月10日(水)
■場所：経団連会館(東京都千代田区)

セイコーエプソン(株)と(株)ルネサステクノロジは，おもにカメラ付き携帯電話の画像表示や画像データ転送向けのシリアル・インターフェース仕様「Mobile Video Interface」の策定を終了し，ライセンス・フリーで公開すると発表した．データ転送速度は，1チャネルあたり最大200Mbps，消費電流は1.4mA．電流駆動型のLVDSを用いることにより，電磁放射ノイズの低減を図っている．

本仕様は，LCDやアプリケーション・プロセッサ，ベース・バンドLSI，カメラ部を接続する際の物理層の電気的仕様を規定したもの．今後はデータ・リンク層の仕様についても順次，策定して公開していく．ピン数を削減できることから，折りたたみ式や回転式など，配線がヒンジ部を通るものに最適とのこと．


Mobile Video Interfaceの使用例

ハッカーの常識的見聞録



今月の常識

今一度，ネットワークを見直そう

広畑 由紀夫

☆ 昨今は非常によく耳にし，また宣伝文句として使われる「セキュリティ」．この春，今一度，見直してみませんか．

この一年，いたるところでセキュリティにかかわる話題が良くも悪くも提供されてきました．ソフトウェアにおいては，信頼していたものがその足元から崩れたり，会社においてはシステムよりも人的なセキュリティがおざなりにされることにより，大問題に発展する事件が起きるなど，より一層の多方面にわたるセキュリティ管理が重要視されるようになっています．

また，Windows Updateなどに代表される自動アップデート機能の利用者の増加などで，新規に見つかったソフトウェア上の不具合などへの対処は以前に比べて楽になり，初心者でも使いこなせる環境が整ってきたかと思います．

さて，今回は，ネットワーク管理では当たり前となってきているアップデートの話ではなく，ちょっと視点を変えてDHCPなどの，最近一般的になってきた自動機能の弊害について考察してみます．

● DHCPの便利さと影

家庭にブロードバンドが普及するにつれて，初心者がほとんど何も設定しなくてもインターネットに接続できる機能が普及してきました．

そのなかの一つにDHCPサーバがあります．現在よく見かけるDHCPサーバは，家庭用ブロードバンド・ルータやモデムなどに搭載され，標準で使用可能になっていることもあり，パソコンやXboxなどのゲーム機器をつなげるとIPアドレスなどの設定を行わなくても，ブロードバンド・ルータの設定だけで使えるという非常に便利なものです．特に一般家庭では，業者が設置する場合はDHCP機能を用いて複数の家庭内LAN上のパソコンやゲーム機がつながるようにすることが多いようです．

● DHCPとは

DHCPを使用しない場合は，個々のIP，ゲートウェイ，DNSなどを手入力で個別に設定することになります．しかしながら，DHCPサーバを使用することにより，DHCPサーバから受け取る情報で個々のコンピュータやゲーム機などはネットワークを利用できるようになります．

DHCPサーバ機能をブロードバンド・ルータやモデムなどがもつことで，個々の機器の設定の手間が非常に低減され，初心者でも簡単に使うことができるようになった点はとても良いことだと思います．

● 現在，筆者が気をつけているDHCP関連の問題

さて，このように非常に便利なDHCP機能ですが，「自動で行えてしまうために社内などでは個人が勝手に持ち込んだPCまで認証してしまう」という弊害が起こりえます．無線LANではMACアドレス登録などの制限の設定を行えますが，有線LANにおけるDHCPサーバ単体でのクライアント認証は不十分ではないかと思われます．

さらに，クライアント認証を導入したとしても，「認証を行い接続を認めたパソコンなどで管理者以外の人間がDHCPが便利だからといって無断でDHCPサーバを立てる」ことにより，ネットワーク自体に不具合が発生することもありました．

● 今後の課題と対処法

もちろんDHCPサーバ自身にMACアドレスなどのクライアント認証機能が併用されるだけでも，他人による無断使用はかなり制限されるかと思います．さらに，ゲートウェイ機器自身にも従来のIPアドレス，サブネット・マスク，ポート制御だけではなく，MACアドレスなども併用した制御が安価に提供されればと考えています．

そのうえでクライアントの使用者が適切な人かどうかを認証するシステムで補うことにより，ネットワークをより安全に維持できるのではないかと思います．

家庭内においては，認証などの必要なレベルが会社内ほど高くはありませんが，多くの家庭内の機器にコンピュータが内蔵され，ネットワークに接続される機器が増えれば増えるほど，使用者の不注意な使用による不具合に自動的に対処できる機能が必要になってくることでしょう．

現状でもっとも簡単な対処法は，ローカルなLAN内のサブネット・マスク境界に含まれるIPアドレスをpingすることで，「使用されないはずのIPが存在するかどうか確認する」，「DHCPサーバの状態監視にて許可した以外の機器が接続されていないかどうかを確認する」といったことを定期的に行うことでしょう．

しかしながら，これらはあくまでも人的な作業として行うため，LAN内のコンピュータに不正な持ち込みがないかどうかを識別するソフトウェアなどの導入も今後は必要になるのかもしれません．

OSにおける脆弱性が悪用されることを防ぐためには，さらなる機器単位でのファイア・ウォール機能などが必要かと思いますが，それらを「利便性を高めるために解除して使用する人も存在する」と仮定して対処することが今後は必要になってくると考えられます．

新製品も数多く発表され，新しい機器への入れ替えなども起こる春ですから，今一度，ネットワークを見直してみてはいかがでしょうか．

ひろはた・ゆきお　OpenLab.

IPパケットの隙間から

IP電話の導入と情報漏洩

祐安 重夫

インターネットの接続回線にADSLを導入したのは，ほぼ2年前のことだった．並行してISDNを引いていたこともあり，回線の干渉などから1Mbpsではそれほど魅力を感じていなかったのだが，8Mbpsのサービスが登場し，しかもISDNとは干渉を起こしにくいということで契約をしてみたのだった．

NTTの交換局からの直線距離は0.9kmなので，条件としては決して悪くはなかった．実際にルータ自体の表示する下りの伝送速度は約5.5Mbpsで，ブロードバンド・スピード・テストのページでチェックすると2Mbpsから2.5Mbpsは出ていたので，こんなものだろうと思っていた．

ADSLはとりあえずつなぎの技術で，これはあくまで個人契約のプロバイダのもの，現在ISDNで接続している会社の独自ドメインは，将来は光という方向で考えていた．ただ，固定IPアドレスを複数持つかたちで光へ移行すると，料金が急に高くなるので，それが安くなるのを待っていたのだ．

しかし，IP電話が使えないかと調べていたら，ADSLを40Mbpsに変更してIP電話を付けても，今のままでIP電話を付けても，料金に大差がないということがわかった．ところで，40Mbpsにすると，どの程度速くなるのだろうか．

2年前，NTTは経路長などを公開していなかったが，今は調べることができ，経路長が1890m，伝送損失が31dBということがわかった．回線業者のシミュレーションによると，12832kbpsから3840kbpsという結果である．まあ，いくらかは速くなりそうではある．

ということで早速契約を申し込んだら，すぐに工事日が決まり，工事日の2日前には新しいADSLルータも届いた．ルータは例によってプライベート・アドレスでDHCPを使用する設定にされていたので，前回と同様まだ生き残っていた古いWindows95マシンを利用してIPアドレスの設定とDHCP機能の停止を行った．これでLANに接続し，必要な設定はとりあえず全部行った．

前回は速度を出すために，アナログ側とディジタル側の両方にフィルタを入れたり，電話ケーブルを短めのツイスト・ペアにし，さらにフェライト・コアを入れて安定性と速度をいくらか稼いだが，今回はもっと良いものがあるかもしれないと秋葉原まで出かけたら，フェライト・コア内蔵のケーブルが出ていた．

この際なのでケーブルを全部交換することにして，設置場所も考慮してケーブル長もできるだけ短くした．壁のモジュラ・プラグからスプリッタまでは30cmという短かさである．

さて，工事の当日，目を覚ましてみると8Mbpsのルータの接続ランプが消えていた．コンピュータ側のsyslogによると，午前9時を少し過ぎたところで，8Mbpsは接続できなくなったようだ．そこで40Mbpsのモデムにつなぎ直してみると，特に問題もなくあっさりとつながってしまった．今度はルータの表示する下りの速度が約7Mbpsである．ISDNと同居しているのだからまあ良いほうだろう．ブロードバンド・スピード・テストでは3Mbpsから3.6Mbpsだから，1Mbps程度しか上がっていないのはいささか期待外れだったが．

ところで，40Mbpsに変更した大きな理由の一つは，IP電話である．ところがうっかりして，変更手続きの際にIP電話をいっしょに申し込むのを忘れてしまったのだ．つながったのだからすぐに手続きをと思ったのだが，何と工事日から10日間は接続できるかどうかのようすを見るための期間となっており，この間はまだ接続が完了していないため，IP電話の申し込みができない．たしかに10日かけて，結局うまくつながらないユーザもいるのだろうが，これにはまいった．

10日たって，やっとIP電話を申し込んだ．いうまでもないが，40Mbpsへの契約変更からIP電話の申し込みまで，すべてWebからオンラインで行った．ルータの送付以外は，連絡も電子メールかWeb上の確認ページからである．IP電話のほうはすぐに手続きが完了し，設定するとすぐに動き出した．試しに軽井沢のS君にひさびさに電話をかけてみた．日本全国3分8円だから，気楽に市外通話がかけられる．

ちょっと気になっていたのは，ネット・ニュースの配送（gnspoolを使用して，定期的にネット・ニュースをローカルなニュース・サーバにかなりの量をまとめて転送している）などの回線を占有する状況で，電話の会話に影響が出ないかどうかという点だったのだが，これも問題はなかった．40MbpsのADSLルータはVoIP機能内蔵であり，それも40Mbpsにした理由だった．

外付けのTA（テレフォニ・アダプタ）とは違い，ルータ内部での帯域制御によって電話に影響がでないようにしてくれるのではないかという期待があったのだ．もっとも，本当にその機能が有効になっているのかは，よくわからないが．

と，ここまではあまり問題はなかったのだが，ADSLの回線業者がアッカだったという，もしかするとたいへんなことになるかもしれない問題が出てきた．どうやらアッカからもプロバイダからもメールや手紙が来ていないところを見ると，最初に情報漏洩が発覚した201人のユーザには入っていなかったようだが，本当にこれだけの人数で済むのかどうかは，今のところわからない．もし情報漏洩がより大がかりなものになっても，500円の商品券程度では納得できそうにない．

すけやす・しげお　インターメディア・アクセス

特集

ようこそ二足歩行ロボット制御の世界へ

最新メカトロ技術と組み込み制御技術の結晶

二足歩行ロボットには，日本が得意とする最新のメカトロニクス技術や組み込み制御技術が集積されています．本誌で紹介してきたマイコン制御，通信制御，さまざまな信号処理，センサ技術，モータ制御，パワー駆動制御など組み込み技術のほとんどが使われています．さらに独自の設計をするためには，機械・機構部品をCAD/CAMを使って自分で製作しなくてはなりません．

この組み込み技術は，産業用，民生用を問わずあらゆる機器を構成するために必要な要素技術です．

そこでまず，二足歩行ロボットを作っている業界全体の大きな流れ，そこで使われている技術の動向をまとめ，具体例としてロボット試作に使われる技術を紹介します．屈伸，二足歩行，股割りなどの動作をさせるためのノウハウを紹介します．

プロローグ



最新メカトロ技術，組み込み制御技術，経験と熟練によるモノつくり技術の集大成

二足歩行ロボットの現状

吉田 幸作

ASIMOに続け…二足歩行ロボットの世界

つい最近までは大学の研究室か，国家プロジェクトの世界の話であった二足歩行ロボットが身近な存在になってきました．大学生が，アルバイトの給料日ごとにサーボ・モータを買い増ししてロボットを組み立てていく…そんな話が現実になっています．二足歩行ロボットの制御技術はみなさんの手が届く技術になったのです．

2002年2月に始まった二足歩行ロボット格闘技大会「ROBO-ONE」は，すでに第5回を数えます．第1回大会では歩くのがやっと，相手と一戦交える前に自爆して果てるものも少なくありませんでした．しかし参加ロボットの技術レベルは目を見張る勢いで急速に向上してきました．倒れても立ち上がる自立機能，倒れるときに衝撃でメカが壊れるのを避ける受身機能など，これが一般の技術レベルかと驚くほどの進歩です．

二足歩行ロボットへの関心は，ごく少数のマニアや技術者だけのものではありません．ホンダのASIMO，ソニーのQRIO（旧SDR3/4）の登場により，ロボットが茶の間でも関心事として注目されるようになりました．

組み込み制御技術のすべてが結集される二足歩行ロボット

図1は二足歩行ロボットに使われている技術を図示したものです．今日，産業界が新しい機器の開発に必要とするあらゆる要素技術が含まれています．

マイコン制御技術（ハード＆ソフト），センサ技術，メカトロ

図1 二足歩行ロボットに使われる要素技術


写真1
HRP-2(川田工業など)

制御技術，パワー駆動技術，通信技術，電源回路など，組み込み制御技術の粋が結集されています．これらの技術のうち，どれ一つが欠けてもうまく歩かせることはできません．

この組み込み制御技術は，自動車，家電製品，携帯電話，産業用機器，航空機など，あらゆる機器を構成するために必要な要素技術です．二足歩行ロボット自体は，まだ残念ながら大きなビジネスになる段階にはありません．しかし，優秀な二足歩行ロボットを開発できる大学や企業は，最先端の組み込み制御技術をもっています．

これらの技術は総合的なものですが，二足歩行ロボットという形をとると，それはごまかしようのない尺度になります．先陣を切ってASIMOを開発したのは自動車メーカのホンダですが，このロボットの完成度はこの会社の組み込み制御技術の水準を示しているといっても過言ではありません．

組み込み制御技術…それはコンピュータおよびネットワーク技術をアメリカにリードされ，製造技術を中国やアジア諸国に追撃されている日本が，いまだに強い競争力を保持している領域です．

日本が二足歩行ロボットの先陣をきったという事実は，このことを如実に示しています．2002年2月14日午前9時30分，ASIMOはニューヨーク証券取引所の取り引き開始ベルを鳴らしました．日本が1歩リードしている二足歩行ロボット技術が，ひょっとすると世界の経済を動かす時代がくるかもしれません．

二足歩行ロボット開発の現状…大学,企業,行政,一般のロボット

表1(pp.46-47)は現在公表されている二足歩行ロボットの仕様をまとめたものです．いずれも企業，大学，行政，特殊法人もしくはその共同によって開発されたロボットです．身長の順に，同じ開発主体のロボットをまとめました．

開発費が数千万円から数億円に上るものも少なくありません．制御基板からメカ，アクチュエータまで，独自に開発したオリジナル部品が多く使われています．使われている部品のほとんどが新規開発で，量産部品でないことがロボットの開発費を押し上げている最大要因です．

ASIMO，QRIO，HOAP-2はそれぞれホンダ，ソニー，富士通という日本を代表する企業が技術の粋を集めて開発したロボットです．

写真1のHRP-2は，川田工業などが開発したヒューマノイド・ロボットです．実物を見る機会が少ないのでなじみはないかもしれませんが身長154cm，アニメの世界から飛び出してきたようなスマートなロボットです．アクチュエータは30自由度，手のグリップや首の動きも可能な本格的なロボットです．レンタル価格も4,000万円であり，なんとASIMOの倍額です．

写真2のASIMOは，テレビCM出演や東京・有明の日本科学未来館，デパートでもおなじみのロボットです．身長120cm，小学生なみの体型とやさしい身のこなしで親しみやすいロボットの人気者になっています．レンタル料も半端ではありません．1日150万円，1年2,000万円です．

COLUMN 1

二足歩行ロボットの定義

ROBO-ONEの大会規定では二足歩行ロボットの条件として，「歩行動作の間に片足が完全に空中にある時間があること」と規定しています．足の裏にローラが付いた構造や，すり足歩行は「二足歩行」とは認めないということでしょう．

写真Aはタミヤの工作基本セット「歩くティラノサウルス」です．たった1,300円の工作キットですが，単3電池1本で尻尾と首を振りながら歩きます．たいへん巧妙にできた工作キットです．

しかし，大会規定では，この構造を「二足歩行」と認めるわけにはゆきません．規約3-1(a)は「ロボットが立った状態で，上から見た足裏の最外周を結ぶ線が左右の足で重ならないこと」と定めています．

また，最近は出場資格にボックス・ダンスを入れています．大会規定は人間と同じ関節自由度をもっていて，人間と同じような歩き方ができるロボットを目指しています．


写真A　歩くティラノサウルス(タミヤ)

表1　おもな二足歩行ロボットの仕様

仕様 ＼ ロボット名		HRP-2（写真1）	ASIMO（写真2）	KOZOH III（写真6）	ながら2（写真5）	QRIO（SDR-4X II）（写真3）	HOAP-2（写真4）	FREEDOM（写真13）
開発主体		川田工業（株），（株）安川電機，清水建設（株），産業技術総合研究所	本田技研工業（株）	ロボス（株）	（社）岐阜県工業会，岐阜県生産情報技術研究所	ソニー（株）	富士通オートメーション（株）	（株）ベストテクノロジー
身長		約154cm	120cm	100cm	約80cm	58cm	50cm	41cm
体重		約58kg（バッテリ含む）	52kg	20kg	約15kg	約7kg	約7kg	2.5kg
関節自由度	首	2自由度	2自由度	3自由度（眼1，首2）	5自由度	4自由度	2自由度	1自由度
	腕（手を含む）	7自由度×2	12自由度（6×2）	10自由度（5×2）	12自由度（6×2）	20自由度（10×2）	10自由度（5×2）	8自由度（4×2）
	腰	2自由度		2自由度	1自由度	2自由度	1自由度	2自由度
	脚	6自由度×2	12自由度（6×2）	12自由度（6×2）	12自由度（6×2）	12自由度（6×2）	12自由度（6×2）	12自由度（6×2）
	合　計	30自由度	26自由度	27自由度	30自由度	38自由度	25自由度	23自由度
制御マイクロプロセッサ（主プロセッサ）		Pentium III 1.26GHz		Pentium III	SH-4（SH7750）	32ビットRISC×3	Pentium III 700MHz相当（外部PC）	メインCPU：SH-2（SH7145F） スレーブCPU：H8/3687F
OS		ART-Linux（運動系），Linux（視覚系）		RT Linux	μITRON	Aperios（独自OS）	RT Linux	HOS（予定）
制御ネットワーク（体内LAN）				LVDS	RS-485	（シリアル）	USB1.1	RS-485 2線式半二重オリジナル・プロトコル
制御用リモコン通信		無線LAN（操作用端末側）	自立型	自立型	自立型	自立型	USB1.1/無線LAN	自律型/Bluetoothによるワイヤレス・コントロール
サーボ・モータ			サーボ・モータ＋ハーモニック減速機＋駆動ユニット	DCサーボ・モータ27個	DCモータ30個	ISA（独自開発）	RC用サーボ・モータ4個（手開閉，首回転，頭うつむき/上向き），他DCブラシレス・サーボ・モータ×21個）	近藤科学KRS2345ICS/PDS-947FET
電　源		ニッケル水素電池DC48V/18Ah	ニッケル水素電池38.4V/10Ah	ニッケル水素電池36V	（制御系）ニッケル水素電池14.4V/3Ah，（駆動系）ニッケル水素電池24V/3Ah	リチウム電池	ニッケル水素電池24V/2150mAh	ニッケル水素電池DC8.4V/1600mAh
センサ	加速度センサ	搭載	搭載	3軸（±2G）	3軸（±2G）	7軸	3軸（±2G）	オプション（JA-30SA32-15）
	ジャイロ・センサ	姿勢角検出センサ	3軸	3軸（±90°/s）	3軸（±90°/s）	3軸	3軸（±60°/s）	オプション（ADXRS300）
	力センサ	脚6軸力センサ，ハンド6軸力センサ	6軸力センサ					
	感圧センサ			4個（足裏センサ×2）		8軸（足底センサ）	8（足底4×2）	オプション（FSR model400）
	測距センサ					3		オプション
	温度センサ				12個（サーミスタ）	28		オプション
	タッチ・センサ					8軸（足底センサ）		オプション
	挟み込みセンサ					18		なし
	グリップ・スイッチ					1		なし
	CCDカメラ	CCDカメラ×3		30万画素×2	C-MOSカメラ×2	11万画素カラーCCD×2	30万画素USB×2（オプション）	オプション（Bluetooth CMOSカメラ）
	音声入力			コンデンサ・マイクロホン×1	マイクロホン（ステレオ）	マイクロホン×7		なし
	音声出力	スピーカ×1		スピーカ×1	スピーカ×1	スピーカ×1		なし
	その他			磁気式3チャネル・エンコーダ（モータ出力軸に取り付け）	両足首に6軸力覚センサを搭載		光学式2相インクリメンタル・エンコーダ×21（DCブラシレス・サーボ・モータに使用）	オプション（AK8970）
販売価格/レンタル価格/非売		レンタル価格 約4000万円	レンタル約2000万円/年	約1200万円	非売品	非売品	オープン価格（約570万円）	50万円（予価）
備考		〒321-3325 栃木県芳賀郡芳賀町芳賀台122-1 川田工業（株）航空・機械事業部ロボティックス部 TEL 028-677-5737 FAX 028-677-5707 E-mail robocraft@kawada.co.jp	〒107-8556 東京都南青山2-1-1 URL http://www.honda.co.jp	〒450-0003 名古屋市中村区名駅南1-5-32タケナカビル501 TEL 052-581-0181 FAX 052-581-3355 E-mail info@robos.co.jp URL http://www.robos.co.jp/		URL http://www.sony.co.jp/QURIO/	〒329-1411 栃木県塩谷郡喜連川町大字鷲宿字西原4415番1 富士通オートメーション（株）エンジニアリング事業統括部第1技術部 第1技術課 TEL 028-686-5475	URL http://www.besttechnology.co.jp/

表1　おもな二足歩行ロボットの仕様（つづき）

仕様 ＼ ロボット名		PINO（OpenPINO）（写真7）	nuvo（ぬーボー）（写真9）	e-nuvo（イーぬーボー）（写真10）	morph3（写真8）	Robovie-R（ロボビー・アール）（写真11）	Robovie-M（ロボビー・エム）（写真12）	WSGH-1（写真14）
開発主体		科学技術振興事業団 ERATO北野共生システムプロジェクト	（株）ゼットエムピー	（株）ゼットエムピー	科学技術振興機構（現在，千葉工業大学未来ロボット技術研究センターにて研究継続中）	（株）国際電気通信基礎技術研究所（ATR）	（株）国際電気通信基礎技術研究所（ATR）・ヴィストン（株）	筆者試作
身長（cm）		70cm	39cm	30cm	38cm	110cm	29cm	41cm
体重（kg）		4.5kg	2.5kg	1.35kg	2.4kg	50kg	1.9kg	2.5kg
関節自由度	首	2自由度			4自由度（眼2含む）	首3自由度 ＋眼4自由度（2×2）		2自由度
	腕（手を含む）	10自由度（5×2）	2自由度（1×2）		12自由度（6×2）	8自由度（4×2）	8自由度（4×2）	6自由度（3×2）
	腰	2自由度			2自由度		2自由度	
	脚	12自由度（6×2）	12自由度（6×2）	12自由度（6×2）	12自由度（6×2）	2自由度3輪 （独立2輪駆動＋従輪1輪）	12自由度（6×2）	12自由度（6×2）
	合計	26自由度	14自由度	12自由度	30自由度	17自由度	22自由度	20自由度
制御マイクロプロセッサ（主プロセッサ）		SH-2（SH7050）		メイン：SH-2E（SH7055F 20MHz） サブ：H8S（HD64F2612 20MHz）	V_R5500（NECエレクトロンデバイス）	Pentium4 1.7GHz	H8	SH-2 （SH7045F）
OS					VxWORKS（WindRiver社）	Linux（RedHat 9.0）	独自方式	
制御ネットワーク（体内LAN）		PWM信号		CANバス	I^2C		シリアル通信	PWM信号
制御用リモコン通信		RS-232-C	PC，テレビ電話（FOMA）によるリアルタイム制御	RS-232-C	自立型	無線LAN（IEEE 802.11b）	ラジコン式	自律型/ Bluetooth通信
サーボ・モータ		双葉電子工業RCサーボ・モータ S5301×14，S9402×12		DCギアド・モータ 定格トルク 390mN・m	DCモータ 30個	Maxon社製	サンワ ERG-VB×14 SPEC-AP2×8	近藤科学 KRS-2346ICS/ PDS-947FET
電源		AC100～120V 8A 入力 DC6V 50A 出力 電源		外部電源 12V 4A	ニカド電池 8.4V/1100mAh	モータ駆動用：鉛蓄電池 12V/12Ah 2個直列， DC駆動用：鉛蓄電池 12V/12Ah 2個直列	ニッケル水素電池 6V/2100mAh	ニカド電池 6.2V/700mAh
センサ	加速度センサ			ZMP-M006GY（オプション） ジャイロ加速度センサ・モジュール	15		2軸	
	ジャイロ・センサ				3			
	力センサ				力覚センサ：24			
	感圧センサ	FSR（圧力センサ）×8		圧力センサ（オプション）				
	測距センサ					赤外線測距センサ×8		
	温度センサ				18			
	タッチ・センサ				8	足元タッチ・センサ×5， 触覚センサ（ON/OFF）		
	挟み込みセンサ							
	グリップスイッチ							
	CCDカメラ	視覚センサ（CMOSセンサ）	27万画素CMOSカメラ	27万画素CMOSカメラ（オプション）	CCDカメラ：2	（前方用）×2		
	音声入力							
	音声出力							
	その他	関節角度 ポテンショメータ		実習カリキュラム教本	電流センサ：31， 電圧検出センサ：1， 赤外線検出センサ：1	関節角度センサ×11，車輪回転角センサ×2，全方位カメラ×1，障害物検知センサ×1，焦電センサ×1	バッテリ電圧センサ×1， 全方位カメラ×1	
販売価格/レンタル価格/非売		304.5万円（Ver.2）～	開発モデル 315万円	68.04万円	非売	約480万円	約39.8万円	
備考		〒108-0073 東京都港区三田1-3-39勝田ビル7F （株）ゼットエムピー TEL 03-5765-6567	〒108-0073 東京都港区三田1-3-39勝田ビル7F TEL 03-5765-6567 http://www.zmp.co.jp/	〒108-0073 東京都港区三田1-3-39勝田ビル7F TEL 03-5765-6567 http://www.zmp.co.jp/	〒275-0016 千葉県習志野市津田沼2-17-1 千葉工業大学 未来ロボット技術研究センター TEL 047-478-0567	〒554-0024 大阪市此花区島屋4-2-7 ヴィストン（株） TEL 06-6467-6601	〒554-0024 大阪市此花区島屋4-2-7 ヴィストン（株） TEL 06-6467-6601	


写真2　ASIMO（本田技研工業）


写真3　QRIO（ソニー）


写真4　HOAP-2（富士通オートメーション）

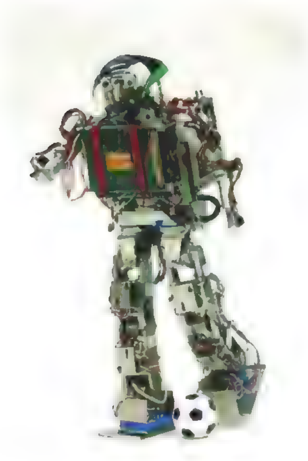

写真5　ながら（岐阜県工業会，岐阜県生産情報技術研究所）


写真6　KOZOH III（ロボス）

ソニーのQRIO（写真3）は，エンターテインメント・ロボットとして時折テレビにも登場しています．以前から「発売も検討」，「その価格は車1台分」という情報も耳にしていました．昨年末の記者発表でQRIO（キュリオ）（旧名：SDR-4X II）と改名，ジョギングやジャンプ・シーンも披露されました．QRIOの技術仕様は開示されていませんが，SDR-4Xから大きなハードウェア上の変更はないようです．昨年のロボットの展示会ROBODEX2003の会場で入手したSDR-4X IIのカタログから仕様を抜粋しました．記者会見では「発売計画はない」との回答でしたが，あのAIBOをヒットさせたソニーがどのような展開を打ち出すのか，興味深く見守っている方は多いでしょう．

HOAP-2（写真4）は，富士通オートメーションから販売されているロボットです．技術情報もよく開示されています．大学および官公庁研究所の開発プラットホームとして出荷されているようです．価格も570万円とお手頃（？），高級車を1台買うつもりになれば個人でも購入可能です．

写真5の「ながら」は，ロボカップ2002のヒューマノイド部門で初代最優秀賞を獲得し注目を集めました．岐阜県工業会をオーガナイザにして地域の企業と生産情報技術研究所，大学などが結集し共同開発したロボットです．開発費は数千万円，制御基板からサーボ・コントローラ，ロボットのメカまで，ほとんどの部品がオリジナルで開発されたようです．

実はこのロボット「ながら」には脚光を浴びた2号機と並行して開発が進められた1号機があります．こちらはOSにLinux，通信系にIEEE1394を採用した野心的な設計で進められましたが，現在は展示室で静かに体を休めているはずです．

KOZOH III（写真6）は，「ながら」の開発に加わったロボスが独自開発の技術を加えて商品化した製品です．研究用プラットホームとして営業を開始しています．

PINO（ピノ）（写真7）およびmorph3（モルフ）（写真8）は，いずれも科学技術振興事業団（現 独立行政法人 科学技術振興機構）の北野共生システムプロジェクトで開発されたロボットです．PINOは事業化，morph3は研究開発ロボットとして千葉工業大学の未来ロボット技術研究センターに継承されています．いずれも文部科学省系研究機関で開発されたロボットです．

写真9のnuvo（ぬーぼー）およびe-nuvo（イーぬーぼー）は，本稿執筆中にゼットエムピーから発表されました．いずれもPINOの技術を活用して開発されました．nuvoは携帯電話FOMAから制御したり，カメラ画像を携帯電話に送ることができます．当初は開発モデルとして315万円で発売されますが，年末には普及モデルがリリースされる予定です．e-nuvo（写真10）は下半身だけの教材モデルです．

写真11は，（株）国際電気通信基礎技術研究所（ATR）で開発されたRobovie-R（ロボビー・アール）をヴィストンが商品化したロボットです．2輪駆動ですから二足歩行ロボットとはいえません．ATRはこの技術をもとに写真12のRobovie-M（ロボビー・エム）をヴィストンと共同開発しました．

Robovie-Mは，れっきとした二足歩行ロボットで，価格も398,000円と個人でも手が届く値段です．ヴィストン社のホームページでたいへんバラエティに富んだ動画が開示されています．

写真13は，ROBO-ONEのきっかけを作ったベストテクノロ


写真7　PINO（科学技術振興機構・北野共生システムプロジェクト，ZMP）


写真8　morph3（科学技術振興機構・北野共生システムプロジェクト，リーディング・エッジ・デザイン，千葉工業大学・未来ロボット技術研究センター）



写真9　nuvo（ZMP）

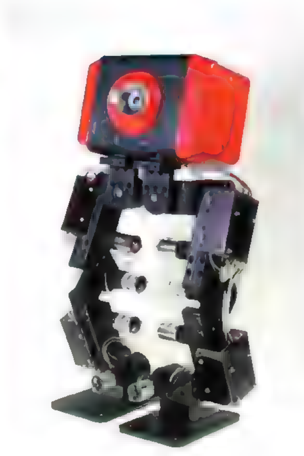
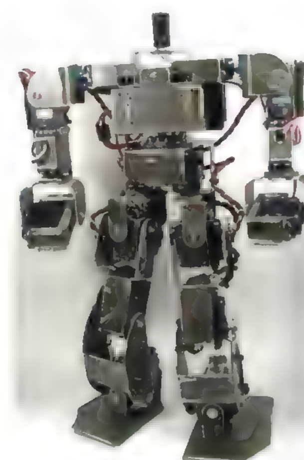

写真10　e-nuvo（ZMP）


写真11　Robovie-R（国際電気通信基礎技術研究所ATR・ヴィストン）


写真12　Robovie-M（国際電気通信基礎技術研究所ATR・ヴィストン）

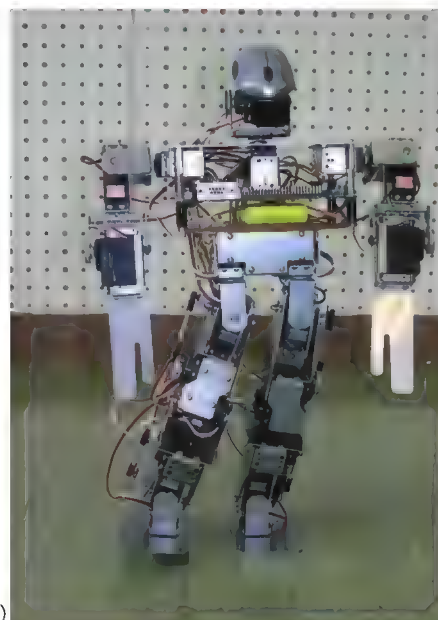

写真13　FREEDUM（ベストテクノロジー）

ジーのFREEDUMです．表1に発売間近のFREEDUMの仕様を示しました．

写真14は，筆者のチームが1余年をかけて完成させた二足歩行ロボットWSGH-1です．メカ部は近藤科学のサーボ・モータKRS-2346ICSおよびイトーレイネツのブラケットを使い，市販されていない部品は金型業者の力を借りました．

制御基板は財団法人ソフトピアジャパンの「IT活用商品開発支援事業」の支援を受けて独自に開発し，制御プログラムも技術チームの力を集めてアルゴリズムの検討からC言語のソース記述まで独自に開発しました．

ここに紹介したロボットの多くは新規に開発した特注部品をふんだんに使っています．時間と開発費を惜しみなくつぎ込まれて完成したロボットです．将来的に事業化される可能性はありますが，現状は研究試作からセミ量産の段階です．ロボットの価格も500万円前後のPINO，HOAP-2からASIMOの数千万円と，工業製品というよりは芸術品に近い存在です．


写真14
WSGH-1（吉田 幸作）

COLUMN2

トヨタの二足歩行ロボット

編集作業を始めたとたんに新しい二足歩行ロボットがトヨタ自動車(株)から発表されました．おもな仕様だけ掲載しておきます．身長：120cm，体重：35kg，首の関節自由度：3，手を含む腕の関節自由度：13，腰の関節自由度：1，脚の関節自由度：12，制御用プロセッサ：Pentium系，OS：RTLinux，制御ネットワーク：体内LAN，サーボ・モータ：29個，電源：6Ahのリチウム・イオン電池，問い合わせ先：TEL 0800-700-7700



二足歩行ロボットが開発試作品から工業製品になるためにはロボット部品の標準化と流通がポイント

筆者はロボット構成部品および構成モジュールが汎用部品として流通し，いつでもどこでも誰にでもロボットが開発できる時代が来ることを期待しています．それは二足歩行ロボットが芸術品から工業製品，大衆商品として流通ルートにのる時代です．

大学や企業，官公庁が開発したロボットのパーツがパーツ単体として流通すれば，二足歩行ロボットはもっと身近な存在になるでしょう．ソニーの開発担当者に「SDR-4のサーボ・モータを部品として外販する計画はありますか」と聞きましたが，答はノーでした．富士通のHOAP-2の場合は「需要があれば部品の外販も考えないわけではありませんが，現状は特注に近いので結構高くなりますよ」という返事でした．

最近，PINO用に開発された部品の一部が市販されています．ポテンショメータ内蔵DCギアド・モータ，モータ・ドライバ・モジュール，ジャイロ加速度センサ・モジュールなどが秋葉原のロボット専門店の店頭に並び始めました．これは新しい動きとして注目に値します．制御用インターフェースにはCANも使われています．

ロボット開発のために開発された部品はノウハウの塊ですから，おいそれとは開示できないかもしれません．しかし，開発されたロボット部品が量産化されて流通し，汎用部品によるロボット開発が可能になれば開発費の大幅なコストダウンが可能になります．そして，開発がいっそう加速されるに違いありません．

表1の各ロボットの制御系は通信仕様が多種多様です．ロボット部品が汎用商品として流通するためには，この制御指示信号の標準化が必要です．

PINOおよびFREEDUMは，これまでの二足歩行ロボットとは大きく流れを変えた製品です．サーボ・モータは市販のRC(ラジコン)サーボ・モータを活用しています．

FREEDUM用にイトーレイネツが開発したブラケットなど機構部品は，部品単体として販売が始まりました．サーボ・モータと機構部品を一式そろえると40万円程で入手できます．

個人にとっては大きな負担ですが，従来の研究投資型のロボットから見ると，画期的な流れです．ROBO-ONE出場選手の多くはこれら市販の汎用部品を最大限活用しています．

PINOおよびFREEDUMの登場により，個人でも二足歩行ロボットが作れる時代がやってきました．まさに今，二足歩行ロボット制御技術は象牙の塔を出たのです．

アマチュア無線の例を引くまでもなく，技術の進歩に一般の人が果たす役割は重要です．数千万円，数億円をかけて企業や大学，官公庁研究所が開発したロボットとROBO-ONEに登場するロボットを同列に論じることは適切ではありません．しかし，二足歩行ロボット開発に一般人が登場したこと，これはロボット開発史上，大きな意味があります．



特集の構成

特集では，二足歩行ロボットを支える要素技術について解説します．すでに紹介したように，これは組み込み制御技術のすべてに及ぶ技術ですから，数十ページで完結できる内容ではありません．

これから二足歩行ロボットの開発を始めようとしたら，どのような技術を用意する必要があるか，ある程度わかる内容にしました．「ロボットなんて関係ない」と考えている方も，日常業務で携わっている自動車，携帯電話，AV機器，産業用FA機器などの技術が幅広く使われていることに驚かれるかもしれません．

技術の総論的な紹介に終わらないように筆者のチームが開発したWSGH-1の技術内容を紹介します．

制御のために「二足歩行ロボット制御基板」を2機種開発しました．SH-2を搭載し，サーボ制御用のPWM信号発生回路も専用ハードウェア(CPLD)で実装しています．その技術内容もくまなく紹介します．

制御プログラムの開発にはイエローソフト社のCコンパイラYCSHを使いました．筆者のチームが開発した屈伸，二足歩行，片足立ち，応援団風のパフォーマンスなどを行うC言語プログラムのソースをすべて紹介します．プログラムにはオフセット補正機能，モーション補間ができるモーション・デバッガも組み込まれています．今後の開発を進めるための強力なツールにもなります．

ロボット開発に必要なものは，まずロマン，
次にまとまったお金と時間と忍耐力，
そして制御のためのハードとソフト，メカトロ技術

よしだ・こうさく

1. 各関節の可動構造と体内LANのしくみを理解する

ロボット制御システムの構成と通信技術

吉田 幸作



ロボットに人間のような動作を行わせるには，人間と同じ関節構造にして，各関節を複合的に動かさなくてはならない．ここでは，その指示を送るための通信系ラインである体内LANの実現方法を検討する．（編集部）

二足歩行ロボットは人間の関節構造を模倣する

図1 はHOAP-2（富士通オートメーション）というロボットの外形図です．身長50cm，重量7kgのボディは，人間と同じ関節構造になっています．首，腕および足の関節部は可動構造になっていて，サーボ・モータで動かすことができます．

図2はHOAP-2の関節自由度数と名称です．図1の外形図と対比させると各関節の可動構造（自由度）がわかります．

たとえば左肩には三つのジョイント（JOINT）がありますが，

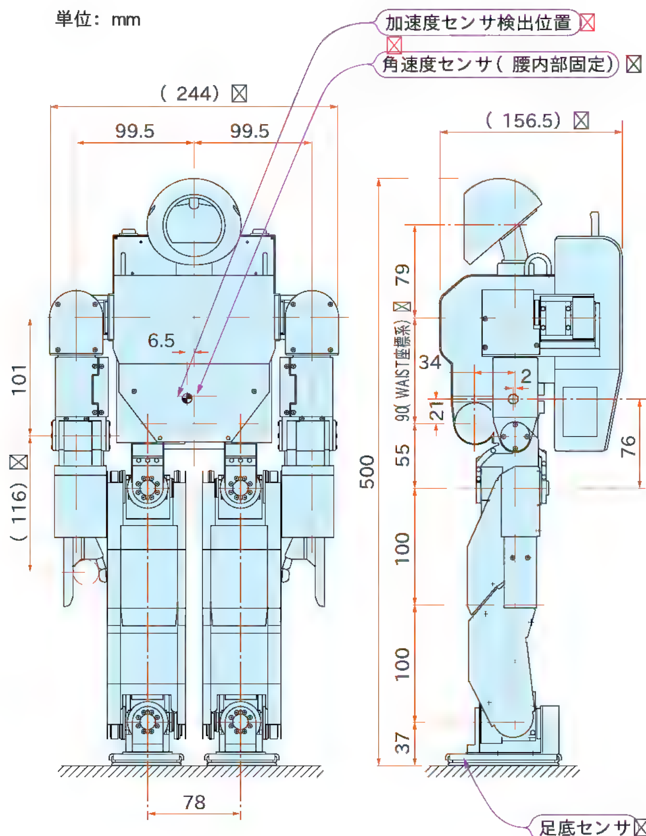


図1[1] HOAP-2（富士通オートメーション）の外形寸法

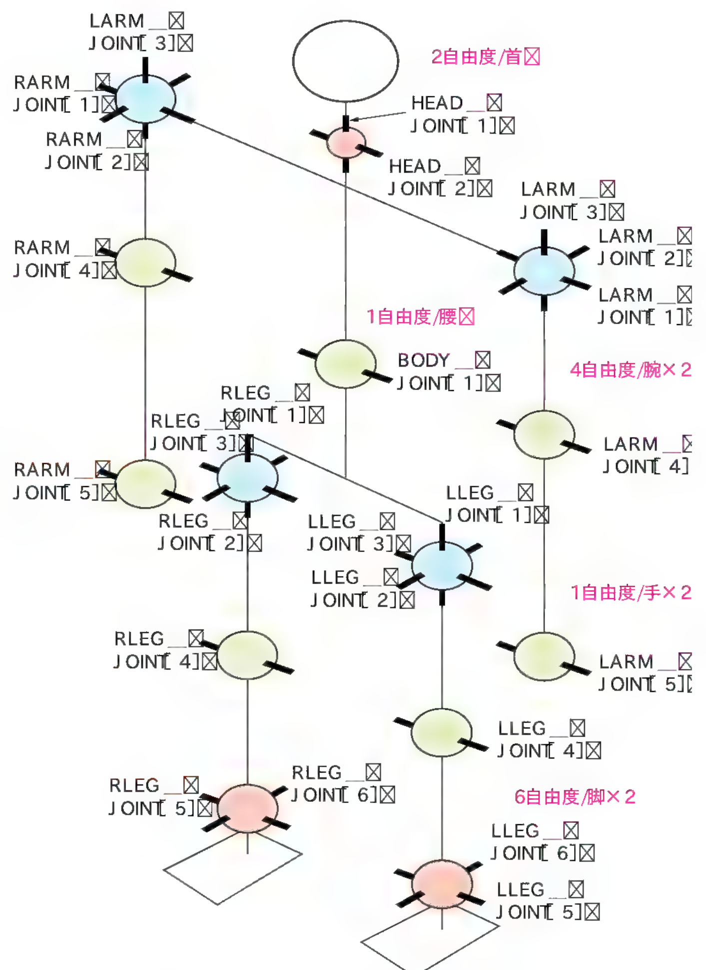


図2[1] HOAP-2の関節自由度数と名称（合計25自由度）

これは図3に示す人間の肩の三つの自由度に対応しています．

前後　LARM_JOINT[1]
左右　LARM_JOINT[2]
軸回転　LARM_JOINT[3]

人間の関節自由度は指の関節まで入れると100程度といわれていますが，基本的な動きを模倣するためには20～30自由度でも十分です．

図4はFREEDUM（ベストテクノロジー）というロボットの関節自由度です．プロローグの表1に主要ロボットの関節自由度と制御ネットワークを示しました．ASIMOからFREEDUMまで，関節自由度には大きな違いはありません．二足歩行ロボットは人間の関節自由度をサーボ・モータなどのアクチュエータで模倣します．自由度の数はそのままアクチュエータの数になります．アクチュエータにはDCモータ，DCブラシレス・モータ，ステッピング・モータが多く使われていますが，ソレノイドなどの電磁アクチュエータが使われることもあります．

集中サーボ方式と分散サーボ方式，さらに局所分散サーボ方式

ロボットの動作を制御するということは，各アクチュエータをどのように動かすかということにつきます．サーボ系のシステム構成方法には図5（a）に示す集中方式と同図（b）の分散方式があります．

集中サーボ方式は軸駆動用モータのサーボ回路を1か所にまとめたシステム構成です．通常は中央制御部の近くに配置します．中央処理部と一体構成にすることもあります．

ロボットの可動部はモータとエンコーダだけで構成して，サーボ回路は安定した場所にまとめて配置します．可動部の重量はサーボ回路の分だけ軽くなり，ロボットのトルク負荷軽減に役立ちます．

しかし，モータとサーボ回路の間を最低でも数本の電源駆動ケーブルとエンコーダ信号線で接続する必要があります．サーボ・モータの数だけケーブル・ハーネスを束ねる必要があるので，サーボ・コントローラの近くではたいへんな太さになります．太いケーブルは実装上の問題だけでなく，ロボットの動きを妨げる要因にもなります．

15年以上前，総重量100kgを超えるロボットの開発で，この集中サーボ方式を採用したことがあります．200WクラスのDCモータをいくつも使いましたが，このケーブルの実装の問題で最後まで悩みました．

図5（b）の分散サーボ方式はモータ，エンコーダ，サーボ回

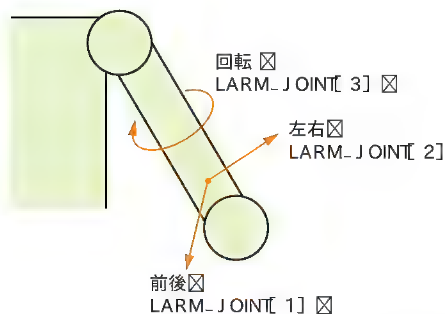


図3　人間の肩の自由度は3…前後，左右，回転

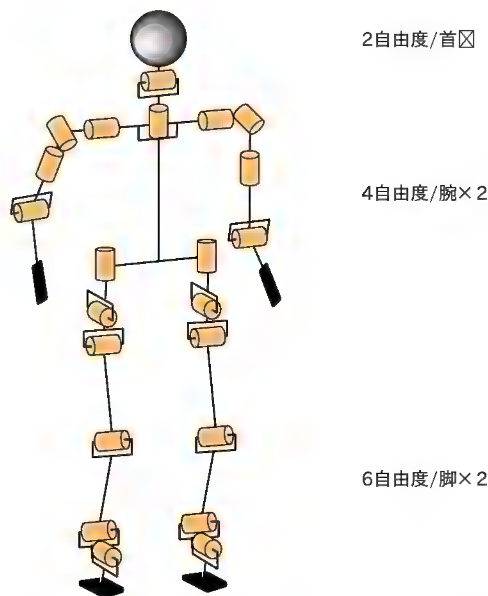
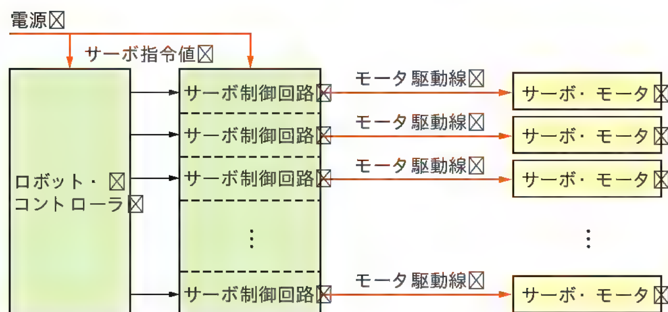
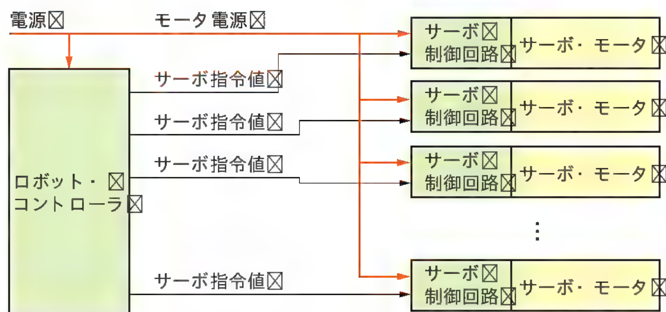


図4[2)]　FREEDUM（ベストテクノロジー）の関節自由度（合計22自由度）



（a）集中サーボ方式



（b）分散サーボ方式

図5　サーボ系のシステム構成方法

路をモジュール化して各可動部に配置した構成です．サーボ指令信号は中央制御部から，駆動電力はロボットの中心部を走る幹線電源ラインから供給します．

電源はサーボ・モータ個別に配線する必要はありません．ロボットの頭から足まで，1本の太い幹線電源ラインを用意して，ここから各モータに給電すれば事は足ります．

分散サーボ方式の欠点は，アクチュエータ部にサーボ回路の重量が加算されることです．二足歩行ロボットの場合，重心は少しでも低くしたほうが安定します．また，腕の先端の重量は大きなトルク負荷となるので，重量は腕の先端ではなく付け根の部分に集中させるほうが安定した機構となります．

分散サーボ方式は，このようなロボットの重心とトルク負荷設計を難しくします．しかし，市販のRC（ラジコン）サーボ・モータはモータ，エンコーダ，サーボ回路がコンパクトにモジュール化され，使いやすくなっています．FREEDUMなどRCサーボ・モータを使ったロボットの多くは，この分散サーボ方式を採用しています．

集中サーボ方式と分散サーボ方式の両方の長所を取り入れた方式が，図6の局所分散サーボ方式です．図のように数個のサーボ・モータの制御回路をひとまとめにして，ロボットの重心のバランスとトルク負荷を考慮して，各部に分散配置します．

この方式はロボットの安定性とケーブル・ハーネスの簡略化をうまくバランスさせることができます．この局所分散サーボ方式は多くのロボットで採用されています．

ロボットのサーボ指示信号を伝える通信技術

現在，多くのロボットは図5（b）の分散サーボ方式，もしくは図6の局所分散サーボ方式を採用しています．いずれの場合も中央制御部から各サーボ回路に制御情報を送る通信系ラインが必要です．人間の神経系の役割を果たすこの通信系は「体内LAN」とも呼ばれます．

プロローグの表1に現行ロボットの体内LANに採用されている通信方式をまとめました．RS-485からI^2C（Inter-IC），USB（Universal Serial Bus），CAN（Controller Area Network），LVDS（Low-Voltage Differencial Signaling），PWM（Pulse Width Modulation）信号などさまざまなシリアル通信方式が使われています．

ロボットの体内LANはサーボの指令値だけでなく，センサ情報の伝達にも使われます．この通信系に求められる条件として，

- 双方向シリアル通信
- 高速
- 高ノイズ耐性

が挙げられます．

コントローラから各サーボに指示値を伝えるだけであれば，1対多の片方向通信でも十分です．しかし，ロボットの各部位にあるセンサ情報を伝達するためには，双方向シリアル通信を行う必要があります．

図7はHOAP-2の制御構成です．HOAP-2はロボット内部のサーボ制御信号およびセンサ情報の伝達にUSBを用いています．コントローラにx86系のマイクロプロセッサを使い，RTLinuxを搭載しています．ハードウェアもWindowsパソコンと同じ

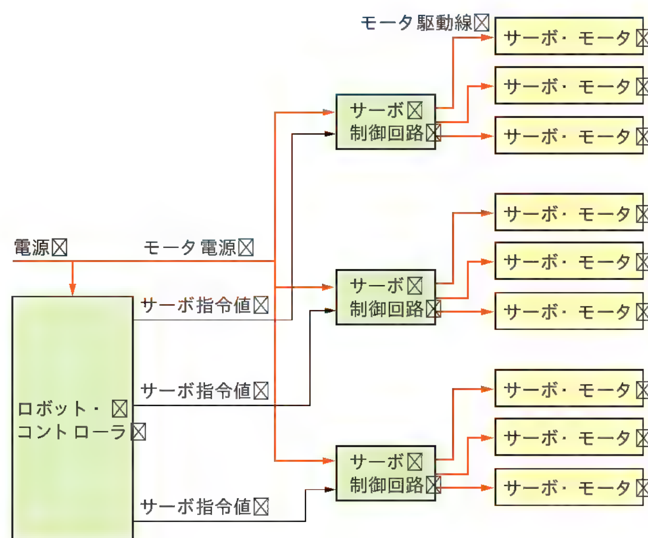


図6　局所分散サーボ方式

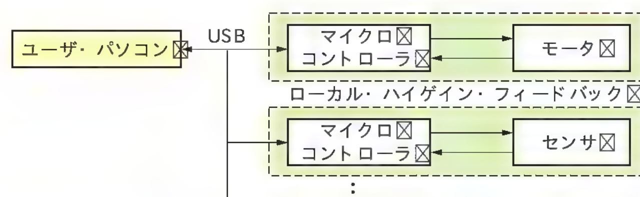
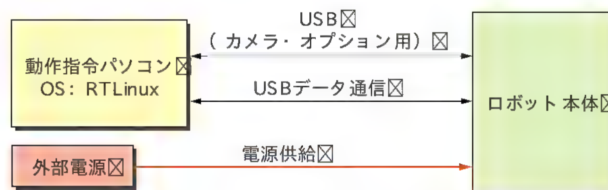
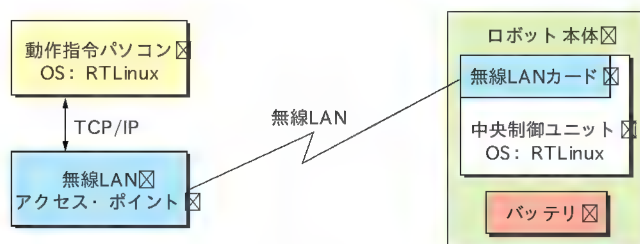


図7[1]　HOAP-2の制御構成



（a）有線モードにおけるシステム構成



（b）無線モードにおけるシステム構成

図8　HOAP-2の通信システム構成

システム構成をとっているので，USBインターフェースを標準で備えています．

HOAP-2は，外部のパソコンから送られてくる指令に従って動くロボットです．図8(p.53)に示すように，有線モードと無線モードの2種類のシステムを構成することができます．有線モードでは，ロボットの体内LANに使われているUSB信号コネクタをパソコンのUSBポートに接続するだけです．

大型のロボットでは，PC/AT互換パソコンのマザー・ボードがコントローラに使われることがあります．このようなシステムではUSBインターフェースが標準で装備されています．これをそのまま通信系に使えばコントローラ側の開発は容易です．

このようなシステムでは，サーボ・コントローラやセンサ側にもUSBインターフェースを実装する必要があります．USBはハードウェアだけでなく，プロトコルも含めて開発する必要があるので，これはちょっとやっかいです．

しかし，最近は「シリアル-USB変換チップ」やUSBインターフェースを備えたワンチップ・マイクロプロセッサも市販されています．これらの部品をじょうずに使いこなせば，サーボ・モータやセンサ側のUSBインターフェースの開発も比較的容易になります．

表1　TIA/EIA(米国電気通信工業会/米国電子工業会)ANSI/TIA/EIA644(LVDS)規格

パラメータ	記号	min	max	単位
差動出力電圧	V_{OD}	247	454	mV
オフセット電圧	V_{OS}	1.125	1.375	V
|V_{OD}の変化量|	ΔV_{OD}		50	|mV|
|V_{OS}の変化量|	ΔV_{OS}		50	|mV|
短絡電流	I_{SA}, I_{SB}		24	|mA|
出力の立ち上がり/立ち下がり時間(200Mbps以上)	t_r/t_f	0.26	1.5	ns
出力の立ち上がり/立ち下がり時間(200Mbps未満)	t_r/t_f	0.26	t_{ui}の30%*	ns
入力電流	I_{IN}		20	|μA|
|スレッショルド電圧|	V_{TH}		100	mV
入力電圧範囲	V_{IN}	0	2.4	V

* t_{ui}は単位間隔，すなわちビット幅

RS-485，CAN，LVDSは，いずれも差動電流駆動型のシリアル・インターフェースです．このうちCANはプロトコルまで規定された通信仕様ですが，RS-485(422)，LVDSは単なる電気信号の仕様で，プロトコルは規定されていません．信号フォーマットとしてマイクロプロセッサ内蔵のUARTを高いビット・レートで使うことも可能です．

RS-485は歴史の古い双方向平衡通信です．しかし，ドライバやレシーバICも10年前とは比べものにならないほど優れたものが出てきています．終端抵抗によるインピーダンス・マッチングをきっちり取れば，10Mbps以上のビット・レートによる通信も可能です．

送受信データ・フォーマットには，制御用のマイクロプロセッサ内蔵のUART(Universal Asynchronous Receiver Transmitter)が使えます．RS-485は歴史が古いので，時代遅れのような印象をもつ人もいるかもしれません．しかし，ロボットの通信系として検討に値する規格ではないでしょうか．

表1はLVDS規格の電気的特性です．400Mbpsクラスの信号伝送仕様として規格化されたものですが，最近はFPGAの高速信号バスとしても採用されています．

図9はAltera社のFPGA"Cyclone"のブロック図です．通常のディジタル入出力端子のほかに高速信号入出力用にLVDSポートが標準で用意されています．

表2はRS-422(485)とLVDSの通信パラメータの比較です．どちらも平衡差動通信ですが，RS-422と比べてLVDSは低電圧，低電流駆動が特徴です．

現在，ロボットの制御で400Mbpsクラスの通信速度は必要ありませんが，低消費電力，耐ノイズ特性を生かしてLVDSを使うメリットはあります．ロボット・コントローラのインターフェース部にFPGAを搭載している場合は，LVDSポートをそのまま使えます．

図10はLVDSのドライバ，およびレシーバ間を100Ωの差動インピーダンス・ラインで接続したときの接続図です．LVDSの伝送路には，必ず100Ωの終端抵抗を付け，信号線は伝送インピーダンスが100Ωのものを使います．

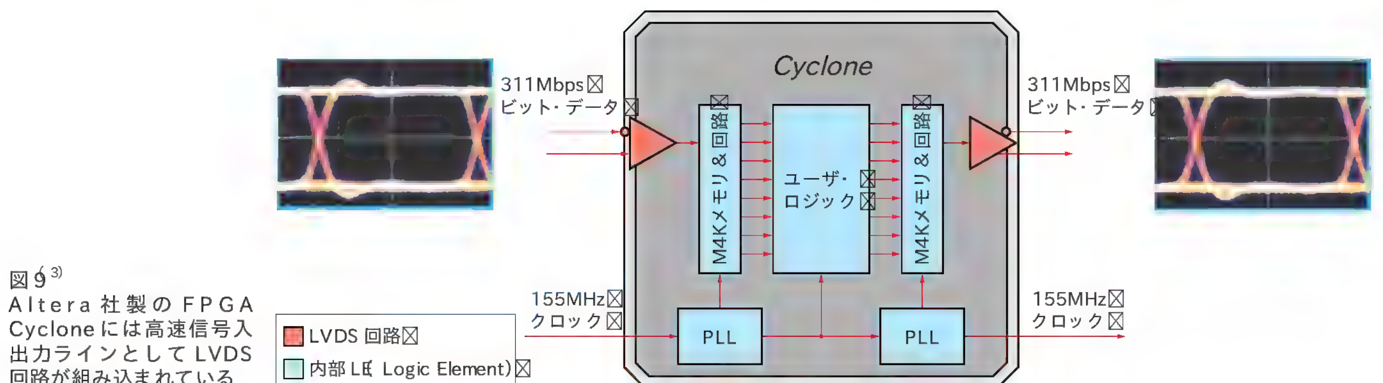


図9[3] Altera社製のFPGA Cycloneには高速信号入出力ラインとしてLVDS回路が組み込まれている

表2[4]　差動通信技術RS-422とLVDSの比較

パラメータ	RS-422	LVDS
差動ドライバ出力電圧	±2～±5V	±250～450mV
レシーバ入力スレッショルド	±200mV	±100mV
データ・レート	<30Mbps	>400Mbps

パラメータ	RS-422	LVDS*
電源電流（クワッド・ドライバ，無負荷，静止状態）	60mA（max）	8.0mA
電源電流（クワッド・レシーバ，無負荷，静止状態）	23mA（max）	15mA（max）
ドライバの伝播遅延	11ns（max）	1.7ns（max）
レシーバの伝播遅延	30ns（max）	2.7ns（max）
パルス・スキュー（ドライバまたはレシーバ）	適用外	400ps（max）

＊表に示したLVDSデバイスはDS90LV047A/048A

このインピーダンス・マッチングは重要です．少しでも不整合があると反射波が発生し，信号の劣化や輻射ノイズの原因になるからです．

図11のように両端にドライバとレシーバを配置すれば双方向半二重通信が行えます．LVDSの通信路は図12に示すように必要に応じていろいろな送受信系を構成することができます．

CANも平衡通信系ですが，プロトコルも含めた規格である点がRS-485やLVDSとは異なります．図13はCANのシステム構成例です．

CANは，自動車の車内LANとして急速に普及している通信規格です．CANを使えば，ロボットと自動車の部品を共通化することもできます．

センサやアクチュエータ部品が流通しているといっても，ロボットだけではまだ十分なマーケットはありません．そこで膨大なマーケットをもっている自動車部品市場と共通の通信仕様を採用すれば，部品の共有化によるコスト・ダウンも期待できます．

CANはプロトコルも含めた仕様なので，ハードウェアとソフトウェアの開発が少したいへんです．自動車用に開発されたCANインターフェース内蔵型マイクロプロセッサがあります．図14はCANインターフェース内蔵H8のTiny，HD64F36057の内部ブロック図です．

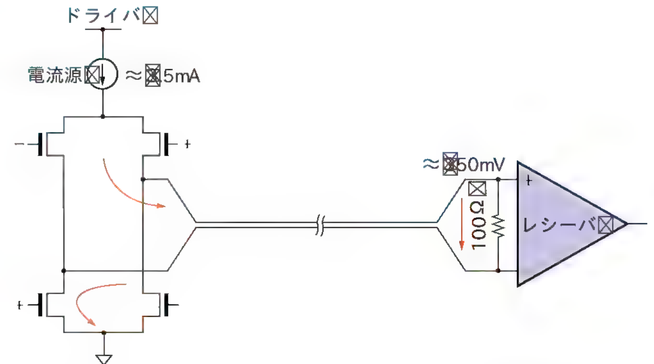


図10　LVDSのドライバおよびレシーバ間を100Ωの差動インピーダンス・ラインで接続したときの略図

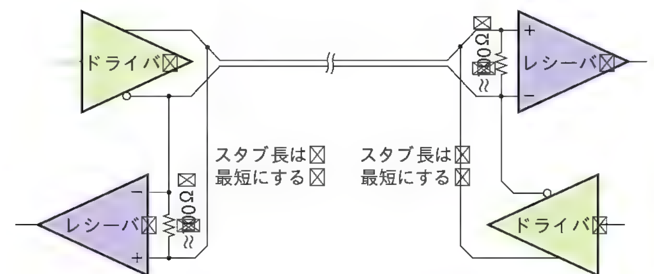
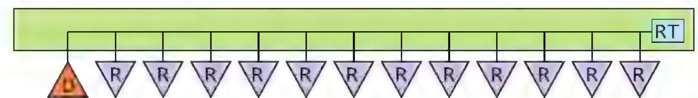
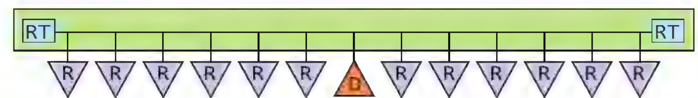


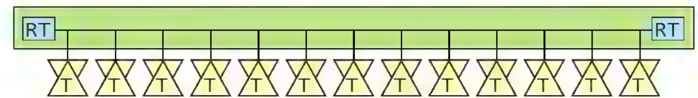
図11　双方向半二重通信のドライバ/レシーバ構成



（a）単一終端方式のマルチドロップ構成



（b）二重終端方式のマルチドロップ構成



（c）マルチポイント構成

図12[4]　LVDSの通信路構成（D：ドライバ，R：レシーバ，T：トランシーバ，RT：終端抵抗）

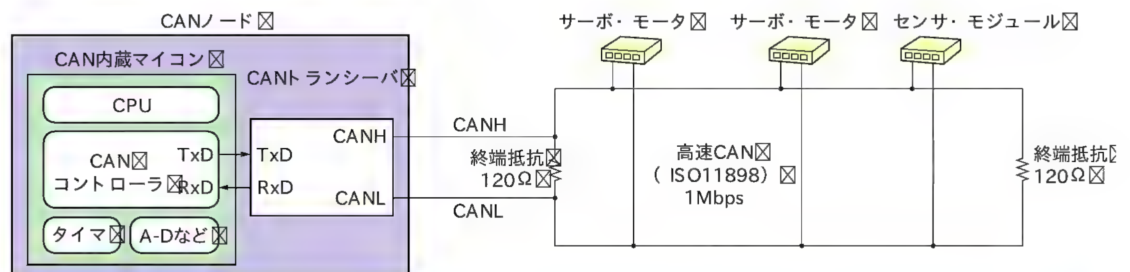


図13　CANの構成

図14[5] CANインターフェース内蔵ワンチップ・マイクロプロセッサ HD64F36057の内部ブロック



写真1 ワイヤレス・ネットワーク・アダプタ〔(株)アイ・オー・データ機器〕



ロボットと外部との通信 …無線LAN，Bluetooth

現在の二足歩行ロボットは，視覚や聴覚情報を駆使して完全に自立動作できる段階にはありません．外部からロボットに動作指示を与えるために無線LANやBluetoothなどの通信手段が使われています．

図8のHOAP-2の場合は，外部の動作指令パソコンとロボット本体との通信に無線LANが使われています．また，ROBO-ONEでもいちばん多く使われているのは無線LANです．

写真1のワイヤレス・ネットワーク・アダプタ〔(株)アイ・オー・データ機器〕は，RS-232-Cインターフェース付きの無線

New Products —— TI，有機ELディスプレイ向けDC-DCコンバータ「TPS65130」を発売
日本テキサス・インスツルメンツ(株)は，バッテリ電源を使用する有機ELディスプレイ向けDC-DCコンバータ「TPS65130」を発売した．出力は－12～15V，効率は最高で89%である．パッケージは4mm角の24ピンQFN．1,000個購入時の単価は$2.95．



写真2　Bluetoothモジュール ZV3001Z(ZEEVO社)



写真3　2.4GHz帯の通信機能がついたCCDカメラと受信モニタ

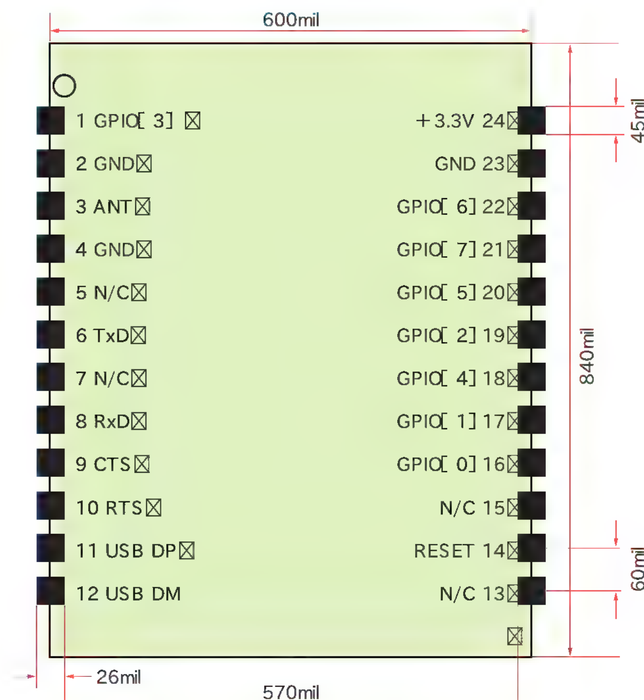


図15[6]　Bluetoothモジュール ZV3001Zのピン配置

LANアダプタです．LANのプロトコルはすべてこのモジュールの中に実装されているので，ユーザはモデム・コマンドを使ってコントローラとロボット間の通信を行うことになります．

無線LANと並んで有望視されているのがBluetoothです．通信距離は標準で7mと無線LANには及びませんが，最近はUARTインターフェースを備えた小型モジュールが市販されています．

写真2はZEEVO社のBluetoothモジュールZV3001Zです．図15はZV3001Zのピン配列です．このモジュールに3.3Vの電源とアンテナを接続すれば，制御用マイクロプロセッサのUARTを介して通信できます．

無線LANよりも小型になるので，機器間の接続性の検証や信頼性の確認が進めば，近距離のロボット・リモコン通信に使われるようになるでしょう．

ロボットの視覚情報センサとしてCCDカメラが使われていますが，この画像情報を外部のコントローラに送るためにも無線通信技術が使われています．

CCDカメラの画像情報をロボットが認識して動作する完全自立型ロボットの実験は始まったばかりです．認識のためのハードウェアもロボットに組み込めるほど小型で軽量なものはまだ実現していません．

このため，ロボットの視覚情報を外部コントローラに送って処理するという方法が一般的です．写真3は2.4GHz帯の通信機能が付いたCCDカメラと受信モニタです．



製作した二足歩行ロボットのシステム構成とRCサーボの制御信号仕様

本誌執筆に先立って二足歩行ロボットWSGH-1を試作しました．図16はWSGH-1のシステム・ブロック図です．

独自に開発した「二足歩行ロボット制御基板」の性能評価のために市販のサーボ・モータと機構部品，一部特注部品を使って開発したのがWSGH-1です．

アクチュエータには市販のRCサーボ・モータを使いました．これはラジコンの制御用アクチュエータとして開発された部品です．当初FREEDUMに使われた近藤科学のPDS-2144FET(トルク13kg/cm)を使いましたが，昨年末にロボット用としてKRS-2346ICS(トルク20kg/cm)が発売されたので，こちらと交換しました．

RCサーボ・モータはモータと減速ギア，位置検出ポテンショメータ，サーボ回路が一体になったモジュール製品です．FREEDUM級のロボットをはじめ，HOAP-2の手のアクチュ

New Products――ARM，ARMv6 ISA向けにLinuxカーネル2.6を最適化するオープンソース・ソフトウェアを提供
英ARM社は，ARMv6 ISA向けにLinuxカーネル2.6を最適化するオープンソース・ソフトウェアの配布を開始した．これにより，ARMv6 ISAの物理キャッシュやASIDS，アトミック・オペレーションといった機能が利用できるようになる．

首 H2 サーボ・モータ☒ KRS-2346ICS
首 H1 サーボ・モータ☒ KRS-2346ICS
腕 B3 サーボ・モータ☒ KRS-2346ICS
腕 B2 サーボ・モータ☒ KRS-2346ICS
腕 B1 サーボ・モータ☒ KRS-2346ICS
腕 A3 サーボ・モータ☒ KRS-2346ICS
腕 A2 サーボ・モータ☒ KRS-2346ICS
腕 A1 サーボ・モータ☒ KRS-2346ICS
脚 R6 サーボ・モータ☒ KRS-2346ICS
脚 R5 サーボ・モータ☒ KRS-2346ICS
脚 R4 サーボ・モータ☒ KRS-2346ICS
脚 R3 サーボ・モータ☒ KRS-2346ICS
脚 R2 サーボ・モータ☒ KRS-2346ICS
脚 R1 サーボ・モータ☒ KRS-2346ICS
脚 L6 サーボ・モータ☒ KRS-2346ICS
脚 L5 サーボ・モータ☒ KRS-2346ICS
脚 L4 サーボ・モータ☒ KRS-2346ICS
脚 L3 サーボ・モータ☒ KRS-2346ICS
脚 L2 サーボ・モータ☒ KRS-2346ICS
脚 L1 サーボ・モータ☒ KRS-2346ICS
二足歩行ロボット 制御基板 ROBO-01A
ニカド電池 6.0V

図16 試作ロボット WSGH-1のシステム構成



写真4 RC（ラジコン）サーボ・モータ PDS-2144FET（左）および KRS-2346ICS（近藤科学）

エータ，PINOなどにも使われています．

写真4は近藤科学のPDS-2144FETおよびKRS-2346ICSです．図17の①に示す4隅の取り付け穴をM3ネジで本体に固定すると，②のサーボ・ホーンが回転します．サーボ・ホーンの回転角度は③のサーボ・リード・コネクタに加えるサーボ指示信号によって決まります．

図18（a）はPDS-2144FETのコネクタ・ピン配列です．3本の配線により，

- サーボ・モータの電源供給

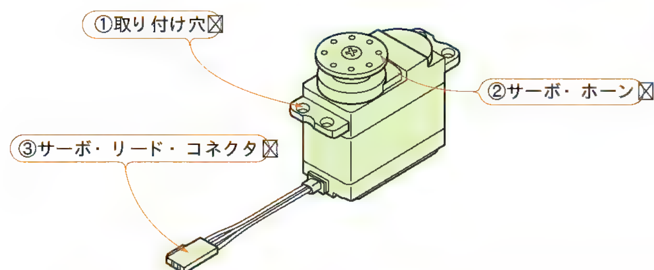


図17[7] RC（ラジコン）サーボ・モータの構成

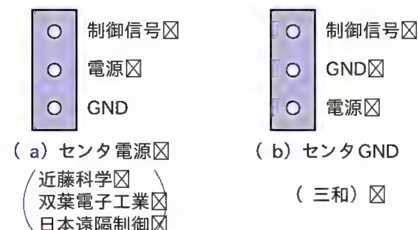


図18 RC（ラジコン）サーボ・モータのコネクタ信号

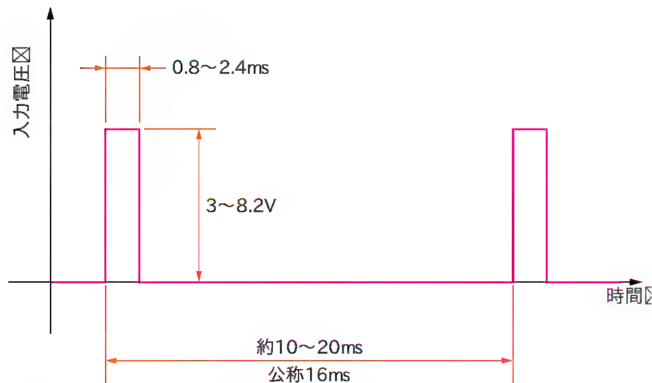


図19　RC(ラジコン)サーボの制御信号

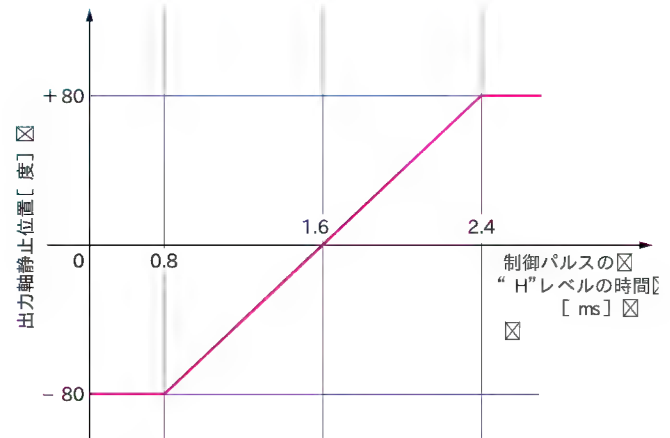


図20　RC(ラジコン)サーボの制御信号と角度制御の関係

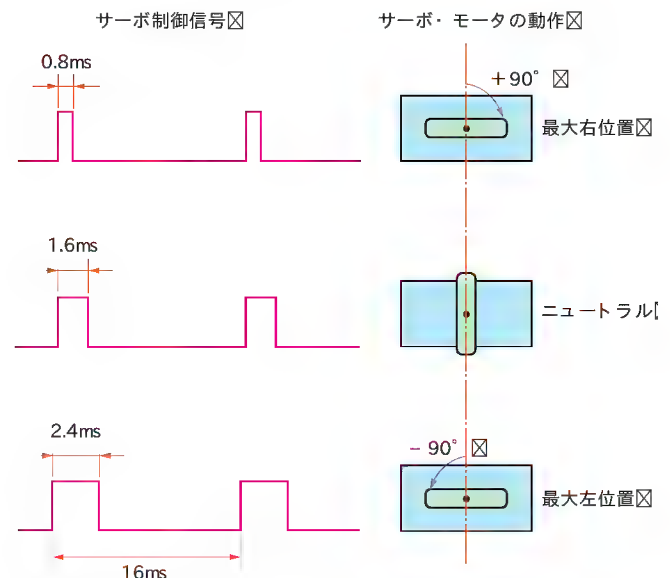


図21　RC(ラジコン)サーボの制御信号とサーボ・モータの動作

● サーボ目標角度情報の伝達

を行います．グラウンド・ラインは共通です．

メーカによって信号配列が一部異なりますが，3ピン・コネクタによりサーボ駆動電源とサーボ角度制御信号を供給する方式は共通です．

図19はRC(ラジコン)サーボの制御信号です．約10ms〜20msインターバルのPWM信号のパルス幅(0.8ms〜2.4ms)によりサーボの目標角度が決まります．

図20に示すように，制御パルス幅を変えるとサーボ出力軸の角度が変化します．実際のサーボ・モータはゼロ点のオフセットがあります．また(パルス幅-制御角度)特性はメーカや機種により異なりますし，理想的な直線ではありません．図21はサーボ角度の制御のようすを図示したものです．

ラジコン用に開発された従来製品の可動範囲は±90度程度です．新たに発売されたKRS-2346ICSはパソコンを使ってサーボ特性の設定をすることにより，±180度の可動範囲設定が可能です．

引用*文献

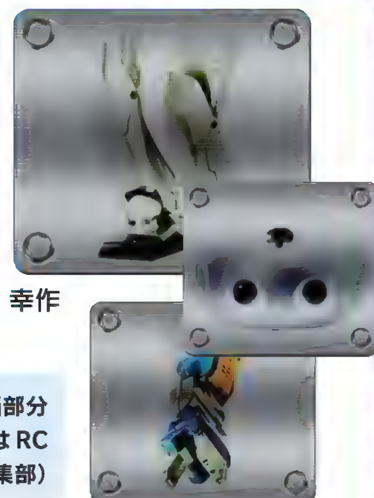
(1)* 富士通オートメーション，小型ヒューマノイドロボットHOPE-2取り扱い説明書，2003年8月
(2)* ベストテクノロジー，FREEDOMカタログ
(3)* アルティマ，Cyclone技術資料
(4)* ナショナルセミコンダクター，LVDS技術資料，2003年1月
(5)* ルネサステクノロジ，HD64F36057データシート
(6)* ZeeVo，ZV3001Z Advance Information
(7)* 近藤科学，RCサーボ・モータ KRS-2346ICSカタログ

よしだ・こうさく

2. RC サーボ制御信号発生回路を CPLD で構成した

二足歩行ロボットの制御回路の設計

吉田 幸作



二足歩行ロボットをうまく歩かせるためには，各モータを動かすための制御指令を送る頭脳部分が重要になる．いかに速く制御信号を出すかが倒れないで歩くためのポイントとなる．ここでは RC サーボ制御信号発生回路を CPLD で構成した制御ボードを開発する．（編集部）

ロボットの頭脳…マイクロプロセッサ

制御部はロボットの頭脳です．ここには制御用マイクロプロセッサが使われています．いちばん重要な役割は歩行アルゴリズムの生成と各関節アクチュエータへの指示信号の生成です．

プロローグで紹介した表1 に，おもな二足歩行ロボットの制御用マイクロプロセッサと搭載 OS を示しました．最近のロボットはこのメイン・コントローラのほかに，サーボや姿勢制御用にサブプロセッサを搭載しています．

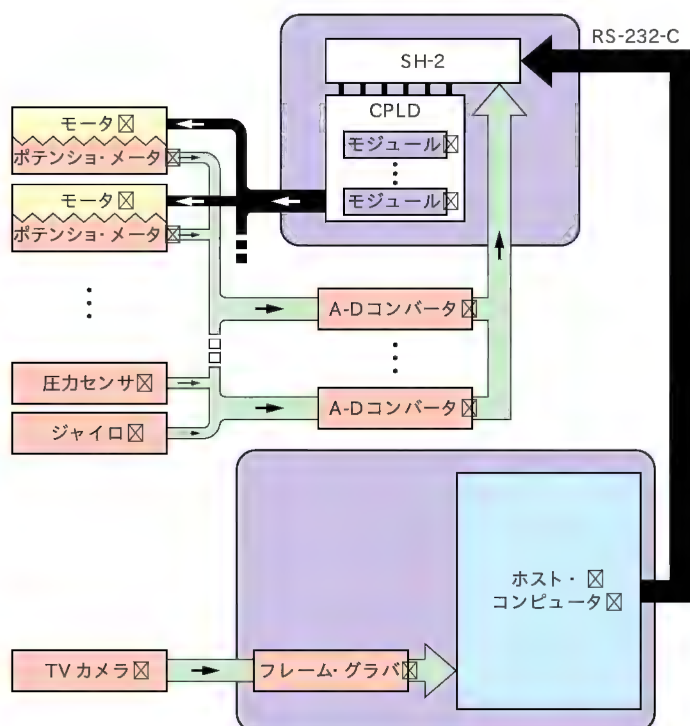


図1[1] PINO のシステム構成

二足歩行ロボットの制御プロセッサに求められる性能はまず第一に処理速度です．ここには Pentium IIIや，64ビットもしくは 32ビット RISC マイクロプロセッサが使われています．

大型の HRP-2，KOZOH III，HOA P-2の CPU は Pentium III，OS は Linux（RTLinux）という組み合わせです．大型のロボットでは実装スペースおよび消費電力の制約が少ないため，このようなシステム構成が可能です．

ホンダの ASIMO は制御チップ名は開示されていませんが，

操作部：ワークステーションおよび携帯コントローラ

という表示があります．

ながら 2，SDR-4X II，FREEDUM，PINO，e-nuvo，WSGH-1は 32ビット RISC，とくに SH 系のマイクロプロセッサが目立ちます．ながら 2には μ ITRON が実装されています．そのほかのロボットは，独自のリアルタイム OS かもしくは未実装です．

FREEDUM はメイン CPU に SH7045F を搭載していますが，OS は未搭載です．

FREEDUM は SH7045F の内蔵タイマを駆使して RC サーボ制御信号を発生させています．サーボ制御信号もカウンタ/タイマの出力ピンをそのまま使っているので，コスト・パフォーマンスもすぐれた制御基板です．

図1 は PINO（OpenPINO）のシステム構成図です．PINO は市販部品を最大限駆使して開発されたロボットです．コントローラには（株）アルファプロジェクト製 SH ボード AP-SH2F-3A が使われています．

サーボ・モータはフタバ電子の RC サーボ・モータ S5301，S3402，S3003を一部改造して使い分けています．

プロローグで紹介した中で唯一，8ビットの CPU H8を使っているのが Robovie-M です．このロボットは，ヴィストン社のホームページにいろいろなアクションの動画が掲載されていますが，8ビット CISCとは思えないような良い動きをしています．

CPU基板：表
LED&GPI
3 2 1 0 power LED
3mm
3mm
2-φ3
VR5500
R2
DSW
N-wire
INT RESET
3mm
3mm

CPU基板：裏
CN1
SYSADコネクタ
(DDコンバータ)
リセット・スイッチ
2.5V
CN2
power

LUKE基板：裏
10
SYSADコネクタ
EDS2508
EDS2508
フラッシュ・メモリ
SROM
OSC
CN
Ethernetコントローラ
クロック
OS
MAX232
10
CN3
パルス・トランス
シリアル コンボ コネクタ
31 1
50mm
50mm

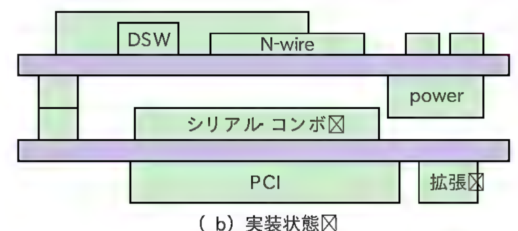
LUKE基板：表
R1, R2, R3
LUKEエンディアン
PLLモード設定
R3 R2 R1
VR5477
LUKE
拡張
CN4
CN2
PCI
R42 R43 R48 R49
Fuse & Link LED
R42, R43, R48, R49
システム・クロック
周波数設定

(a) CPU基板とLUKE基板の部品レイアウト

小さな基板に大きな処理能力…Morph3のコントローラ

プロローグで紹介したロボットの中ではいちばん小粒ですが，大きな処理能力をもっているのがMorph3のコントローラです．図2に示すように50×50mmの基板両面に部品を実装し，さらにそれを2枚重ね合わせた構造です．

V_R5500はNECが開発した64ビットRISCプロセッサで，



(b) 実装状態

図2[2] Morph3に使われているVR5500-ATOM(シマフジ電機)の外形

表1[2] VR5500-ATOMの主要な仕様（シマフジ電機（株））

▶ V_R5500プロセッサ（max 400MHz）
▶ システム・バス周波数：100MHz/83MHz/75MHz/66MHzから選択可能（抵抗オプション）
▶ CPU内部動作周波数：システム・バス周波数の2/2.5/3/3.5/4/4.5/5/5.5倍から選択可能（DIPスイッチ）
▶ 64Mバイト　SDRAMメイン・メモリ（256Mビット×2個使用）
▶ 16Mバイト　フラッシュ・メモリ（インテル製）
▶ GPIO制御LED4個搭載（入力としても使用可能）
▶ リセット・スイッチ，割り込みスイッチを搭載
▶ 周辺機能コネクタ（シリアル・コンボ・コネクタ）：31ピン
● 16550互換シリアル（1チャネル）
● I²C（チャネル-Fast）
● USBホスト（1チャネル）
● AC97 CODECインターフェース（1チャネル）
● Ethernet（1チャネル）
※周辺機能コネクタは，LUKE基板裏面に実装（標準）
▶ 拡張バス・コネクタ
● PCIコネクタ：100ピン
● 周辺機能（GPIO，USB，AC97，UART，I²C）：60ピン
※拡張バス・コネクタは，LUKE基板表面に実装（標準構成では実装なし）
▶ 電源回路　3.3V供給，ボード内で2.5V，1.5Vを生成．
▶ 消費電力　4.5W（typ.）
▶ 外形　50mm×50mm×20mm

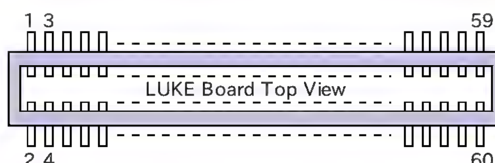
表2[2]
VR5500-ATOMの周辺機能コネクタの信号配列

ピン番号	信号名	I/O	ピン番号	信号名	I/O
1	FG	–	17	ACSYNC	OUT
2	FG	–	18	ACDOUT	OUT
3	RDN	IN	19	ACRST	OUT
4	RDP	IN	20	ACDIN	IN
5	TDN	OUT	21	ACBCLK	OUT
6	TDP	OUT	22	GND	–
7	GND	–	23	UOCIO	IN
8	GND	–	24	UPONO	OUT
9	3.3V	–	25	UDNO	IN/OUT
10	3.3V	–	26	UDPO	IN/OUT
11	GND	–	27	GND	–
12	GND	–	28	GND	–
13	R_XD0	IN	29	SCK	IN/OUT
14	T_XD0	OUT	30	SDA	IN/OUT
15	GND	–	31	GND	–
16	GND	–	–	–	–

注1：3.3V電源の電流最大値は500mA（2本合計）

注2：RxD0，TxD0のインターフェース・レベルはRS-232Cレベル

表3[2] VR5500-ATOMの拡張コネクタ（LUKE-CN4）の信号配列

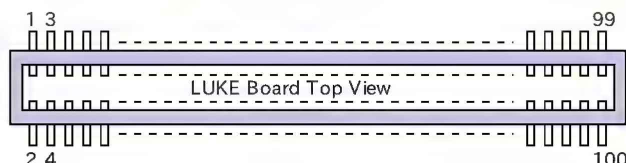


使用コネクタ：
①基板側：日本航空電子製　WR-60P-VF60-1：
②勘合コネクタ：日本航空電子製　WR-60S-VFH30-1

ピン	信号名	ピン	信号名	ピン	信号名	ピン	信号名
1	3.3V	2	3.3V	31	UPON0	32	UOCI0
3	3.3V	4	3.3V	33	GND	34	GND
5	TxD0	6	RxD0	35	UDP1	36	UDN1
7	DCD1	8	DSR1	37	UPON1	38	UOCI1
9	RxD1	10	RTS1	39	GND	40	GND
11	TxD1	12	CTS1	41	UDP2	42	UDN2
13	DTR1	14	RI1	43	GND	44	GND
15	GND	16	GND	45	GPIO00	46	GPIO01
17	SDA	18	SCK	47	GPIO02	48	GPIO03
19	GND	20	GND	49	GPIO04	50	GPIO05
21	ACBCLK	22	ACDOUT	51	GPIO06	52	GPIO07
23	ACSYNC	24	ACDIN	53	GPIO08	54	GPIO09
25	ACRESET	26	GND	55	GPIO10	56	GPIO11
27	GND	28	GND	57	GPIO31	58	RESET
29	UDP0	30	UDN0	59	GND	60	GND

注：UART0，UART1の信号は3.3V CMOSレベル

表4[2] VR5500-ATOMのPCIコネクタ（LUKE-CN2）の信号配列



使用コネクタ：
①基板側：日本航空電子製　WR-100P-VF60-1：
②勘合コネクタ：日本航空電子製　WR-100S-VFH30-1

ピン	信号名	ピン	信号名	ピン	信号名	ピン	信号名
1	3.3V	2	3.3V	51	CBE0	52	CBE1
3	3.3V	4	3.3V	53	CBE2	54	CBE3
5	3.3V	6	3.3V	55	GND	56	GND
7	3.3V	8	3.3V	57	PAR	58	DEVSEL
9	3.3V	10	3.3V	59	FRAME	60	IRDY
11	PAD30	12	PAD31	61	TRDY	62	STOP
13	PAD28	14	PAD29	63	LOCK	64	PERR
15	PAD26	16	PAD27	65	SERR	66	GND
17	PAD24	18	PAD25	67	GND	68	GND
19	GND	20	GND	69	GND	70	INTB
21	PAD22	22	PAD23	71	INTC	72	INTD
23	PAD20	24	PAD21	73	GND	74	GND
25	PAD18	26	PAD19	75	GND	76	REQ1
27	PAD16	28	PAD17	77	REQ2	78	REQ3
29	GND	30	GND	79	GND	80	GND
31	PAD14	32	PAD15	81	GND	82	GNT1
33	PAD12	34	PAD13	83	GNT2	84	GNT3
35	PAD10	36	PAD11	85	GND	86	GND
37	PAD8	38	PAD9	87	PCIRESET	88	GND
39	GND	40	GND	89	GND	90	GND
41	PAD6	42	PAD7	91	PCICLK1	92	GND
43	PAD4	44	PAD5	93	GND	94	GND
45	PAD2	46	PAD3	95	PCICLK2	96	GND
47	PAD0	48	PAD1	97	GND	98	GND
49	GND	50	GND	99	GND	100	GND

Information —— ACCESSの情報家電向けブラウザ「NetFront」がNTTドコモのFOMA 900iシリーズ全5機種に搭載
（株）ACCESSの情報家電向けブラウザ NetFrontがNTTドコモの携帯電話であるFOMA 900iシリーズの全5機種に搭載される．本ブラウザは，HTMLメールにも対応．メールの文字を点滅させたり，音楽データや画像データの受信などが行える．

MIPS系の命令コードを実行します．表1はVR5500-ATOMの仕様ですが，

- CPUクロック　400MHz（max）
- 64Mバイト SDRAM＋16Mバイト・フラッシュ・メモリ
- PCIバス・ブリッジ搭載

と，さながら「手のひらに乗るワークステーション」の感があります．

VR5500-ATOMの周辺機能コネクタには表2に示すように，

- UARTシリアル・ポート
- I^2C
- USBホスト
- AC97 CODEC
- Ethernet

と豊富なシリアル通信機能が装備されています．

さらに拡張仕様として拡張コネクタとPCIコネクタを備えています．拡張コネクタには表3に示すように，

- USBホスト（2チャネル）
- USBファンクション（1チャネル）
- UART（2チャネル）
- AC97 CODEC
- I^2C
- GPIO（汎用入出力ポート）

が接続されています．

また，PCIコネクタには表4に示すPCIバス信号が出力されます．V_R5500はT-Engineにも対応しています．拡張モジュールの追加およびT-Engineミドルウェアの導入により，音声認識，画像処理，マルチメディア処理機能の拡充が可能です．

図3はVR5500-ATOMのハードウェア構成（ブロック図），図4はそのメモリ・マップです．Morph3に組み込まれたV_R5500はリアルタイムOS　VxWORKS（ウィンドリバー社）を搭載しています．

Morph3は研究用のロボットですが，全長38cm，重量2.4kg

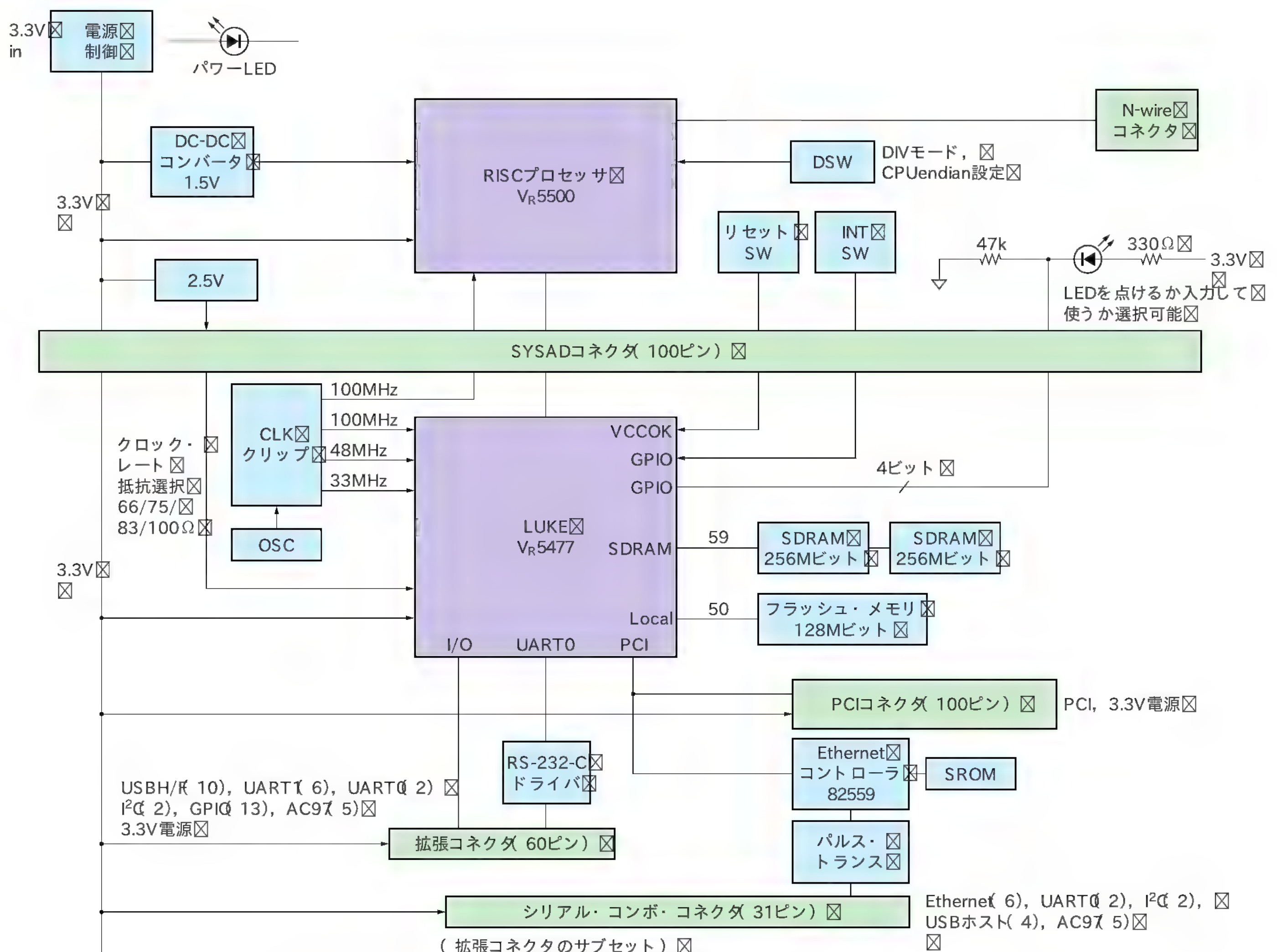


図3[2]　VR5500-ATOMのブロック図

物理アドレス☒　RESET後☒

0x1FFFFFFF

BOOTCS☒
2Mバイト☒

0x1FC00000

INTCS☒
2Mバイト☒

0x1FA00000

0x00000000

初期化後☒

0x1FFFFFFF

BOOTCS☒
フラッシュ・メモリ☒
16Mバイト☒

0x1F000000

LCS2☒
未使用☒

LCS1☒
未使用☒

LCS0☒
未使用☒

INTCS☒
2Mバイト☒

0x1BA00000

IOPCI WIN1☒
16Mバイト☒

0x16000000

IOPCI WIN0☒
16Mバイト☒

0x15000000

EXTPCI WIN1☒
16Mバイト☒

0x14000000

EXTPCI WIN0☒
64Mバイト☒

0x10000000

SDRAM BANK23☒
未使用☒

0x04000000

SDRAM BANK01☒
64Mバイト☒

0x00000000

図4[2]　VR5500-ATOMのメモリ・マップ設定例

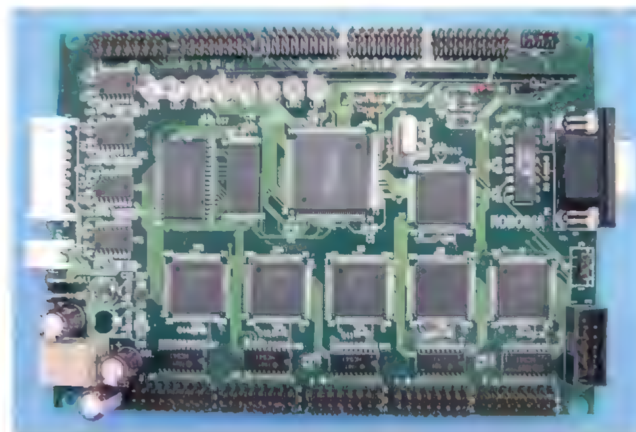


写真1　二足歩行ロボット制御基板 ROBO-01A

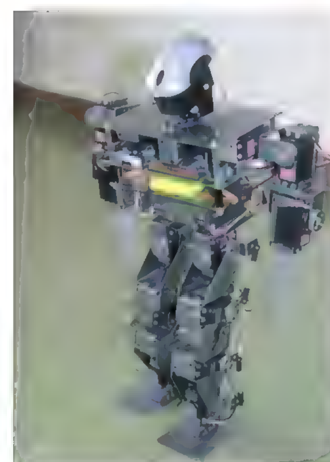


写真2
製作した二足歩行ロボット
WSGH-1

の小さなボディに最新技術の粋が組み込まれています．「小型ロボットを極めれば大型はおのずと可能」という開発者のポリシが感じられます．

Morph3は研究性の高いロボットですが，この中枢部に使われているVR5500-ATOMはシマフジ電機(株)の製品です．定価15万円ですが，なんとロボット王国の楽天モールではコンボ・ボード付き128,000円で販売されています．

二足歩行ロボットの制御基板の開発…RCサーボ制御信号発生回路をCPLDで構成

本誌執筆に先立って二足歩行ロボット制御基板ROBO-01を開発しました．二足歩行ロボット制御に特化した制御基板があればロボット開発が容易になると考えたからです．部品を実装しただけの試作段階のROBO-01は，2003年4月に開催されたROBODEX2003に参考出展しました．

二足歩行ロボットの開発は身近になったとはいえ，メカの製作から制御回路の製作，そしてソフトウェアの開発と，幅広い知識と熟練，そして製作のための作業工程が必要です．二足歩行ロボットを自分の手でと思っても，すべての人がメカから電子回路，ソフトウェアの専門家であるわけではありません．

イトーレイネツ社からサーボ・ブラケットと機構部品が発売されたことにより，メカの知識や工作機械がない人でもロボットの機構一式を手に入れることが可能になりました．

ロボット制御のハードウェアについても電源回路からサーボ・コネクタの結線までワンタッチでできる基板があれば便利です．ハードウェアに自信がない方はもちろんのこと，自信がある人でもオール・イン・ワンの制御基板があれば配線作業の時間を大幅に節約できます．

オリジナルで設計するのであれば，PWM発生回路はCPLDで実装して制御効率を向上させよう，RCサーボ・モータだけでなくDCギアド・モータも用意しようと，思い入れを込めて設計したのが二足歩行ロボット制御基板ROBO-01です．

CPLDは使い慣れたラティスセミコンダクター社のM4A5-128/64を4個実装し，1個あたり8チャネルのPWM発生回路の実装は容易と踏んで回路設計と基板設計を先行させました．

ロジック設計は基板ができてからゆっくり手をつけようと考えてあと回しにしましたが，いざ論理設計を始めてみると，10ビット分解能のPWMコントローラはフィッタをどのよう

New Products――テクノマセマティカル，携帯電話用CPU上でも動くH.264規格の動画再生ソフトウェアを発売
(株)テクノマセマティカルは，H.264/MPEG-4 AVC規格の動画再生用ソフトウェアを開発し，受注を開始した．本ソフトウェアは，同社が開発したアルゴリズム「DMNA」を応用し，携帯電話内のCPUだけで動作するよう，軽量化が図られている．

表5　二足歩行ロボット制御基板ROBO-01Aのハードウェア仕様

マイクロプロセッサ&メモリ	マイクロプロセッサ	SH-2（SH7045F）28MHz（水晶振動子7MHz）	
	メモリ	内蔵フラッシュ・メモリ	256Kバイト
		内蔵RAM	4Kバイト
		外部SRAM	512Kバイト
		外部フラッシュ・メモリ	512Kバイト
	メモリ・マップ	内蔵ROM	00000000H ～ 0003FFFFH
		内蔵RAM	FFFFF000H ～ FFFFFFFFH
		外部SRAM	00400000H ～ 0047FFFFH
		外部フラッシュ・メモリ	00800000H ～ 0087FFFFH
		PWM制御レジスタ（CPLD）	00C00000H ～ 00C000FFH
	フラッシュ・メモリ書き換え許容回数	SH7045F内蔵フラッシュ・メモリ	100回（保証）　1,000回
		外部フラッシュ・メモリ	10万回（保証）
RCサーボ制御回路	回路方式	CPLD（M4A5128/64）による専用ハードウェア搭載	
	制御チャネル数	36チャネル　（ROBO-01A）	
	内臓タイマによるPWM発生回路	16チャネル	
周辺インターフェース	A-Dコンバータ	8チャネル10ビット	
	DCモータ・ブリッジ駆動回路	4回路	
	ディジタル入力	8ビット（A-Dコンバータと兼用）	
	ディジタル入出力	36ビット（うち16ビットは内蔵タイマ回路と兼用）	
	RS-232-Cインターフェース	2チャネル	
電源	基板供給電源	3.0～6.0V	
	CPUおよびロジック回路	基板供給電源よりスイッチング回路により5.0Vを発生	
	RCサーボ電源	基板供給電源を使用	
	DCモータ電源	基板供給電源もしくは別途外部電源1.5～6.5V	
	基板供給電源はそのままRCサーボの電源として使うので，使うRCサーボに合わせる．単3ニカド電池5本を推奨． CPUおよびロジック部電源は，基板供給電源をダイオードでレベル・ダウンした後，ステップ・アップ・スイッチング・レギュレータ回路で5.0Vにする		
開発ツール	市販の各社Cコンパイラを利用できる．メーカ純正Cコンパイラ，イエローソフトCコンパイラ（YCSH），GCCなどを利用可能． イエローソフトCコンパイラ（YCSH）用のスタート・アップ・プログラムおよびダウンローダを供給		

にくふうしても1個あたり6チャネル，トータル24チャネルしか入りません．

総計24チャネルでも二足歩行ロボットの通常機能の制御には不自由しません．しかし首および視覚カメラの駆動，手先や指の制御まで考えると十分とはいえません．

とりあえずROBO-01でソフトウェア開発を進め，並行してCPLDを6個搭載したROBO-01Aの開発を進めました．この基板は回路構成も大幅に見直しました．開発資金は財団法人ソフトピアジャパンの「IT活用商品開発支援事業」の支援を受けました．

こうして完成したのが写真1の基板ROBO-01Aです．この基板をコントローラにした試作ロボットWSGH-1（写真2）は2，3か月で歩き始めました．

完成した二足歩行制御基板の構成

図5（p.66）は制御基板ROBO-01Aの回路ブロック図，表5はその仕様書です．105×148mmのガラス・エポキシ4層基板に表5の機能を凝縮しました．

制御には32ビットRISCプロセッサSH7045F（ルネサステクノロジ）を使いました．このチップはSH-2シリーズの原点ともいえるプロセッサです．

図6（p.67）はSH7045Fの内部ブロック図です．クロック周波数は28MHzで，

- 256Kバイト・フラッシュ・メモリ
- 4KバイトSRAM
- 8チャネル10ビットA-Dコンバータ
- 2チャネル・シリアル通信インターフェース
- マルチファンクション・タイマ・ユニット
- 2チャネル・コンペアマッチ・タイマ
- ウォッチドッグ・タイマ
- ディジタル入出力ポート（ROBO-01Aで利用可能なポートは36ビット）

など，豊富な周辺入出力機能を備えています．

SH7045Fは32ビットRISCプロセッサなので，内部バスは当然32ビットですが，外部データ・バスは8/16/32ビットのうちから選択できます．

ROBO-01Aは，図5に示すようにシンプルな8ビット・バスを選択しました．データ・バス配線を少しでも簡略にして基板をシンプルにしたかったのが第一の理由です．プログラムのほとんどは内蔵フラッシュ・メモリ上で動くので，実行速度に大きな影響はありません．

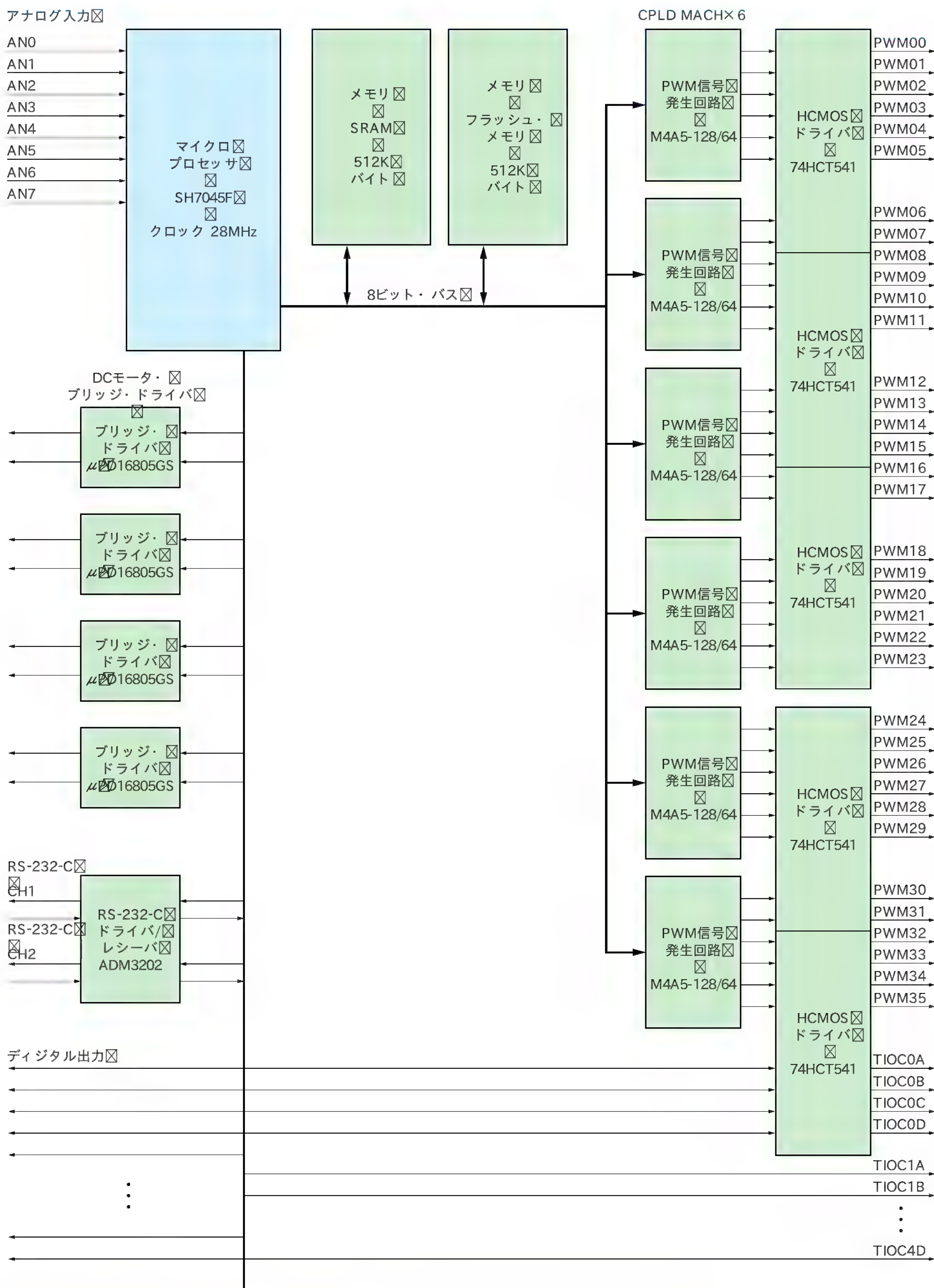
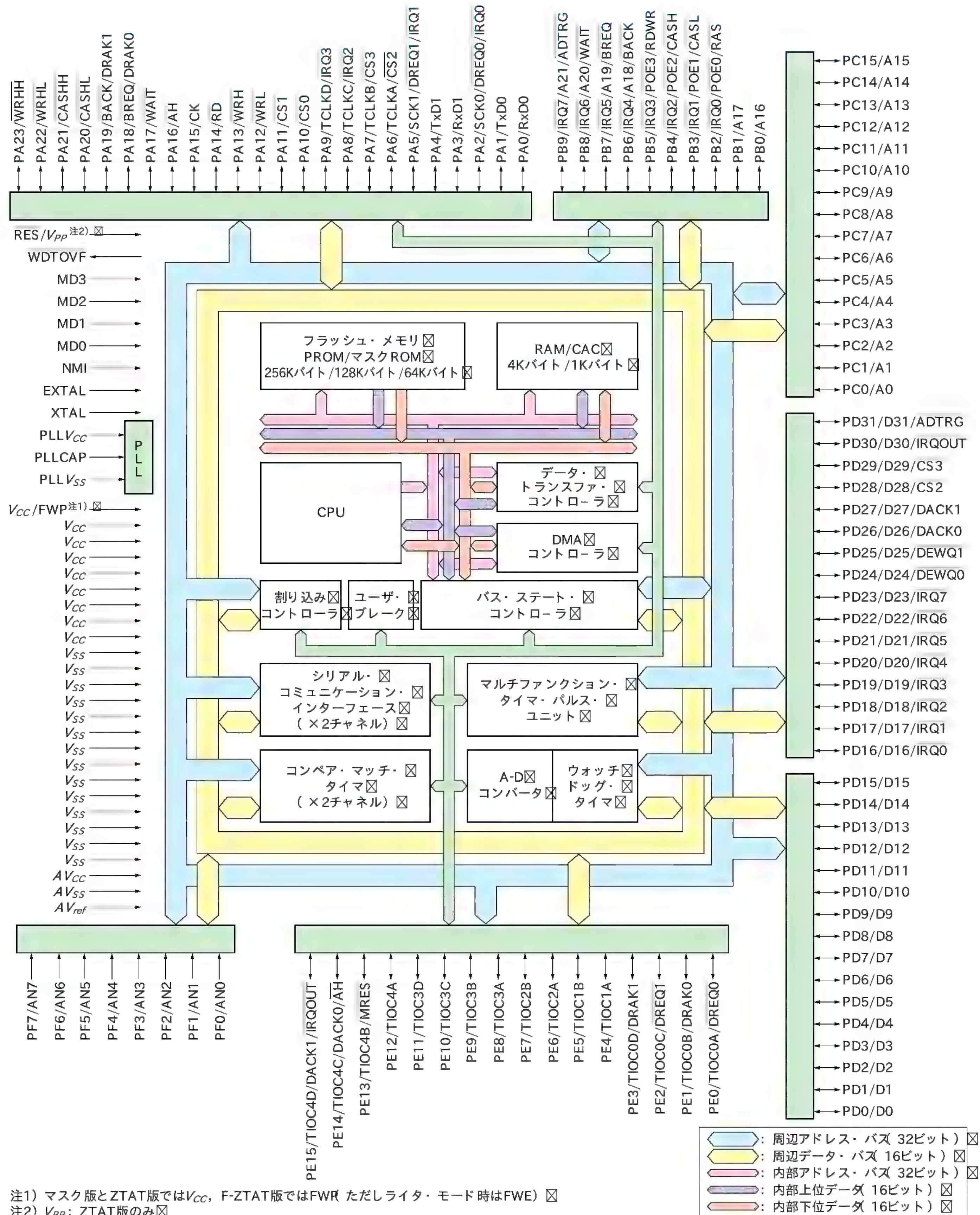


図5 二足歩行ロボット制御基板ROBO-01Aの回路構成



注1)マスク版とZTAT版ではV_{CC},F-ZTAT版ではFWP(ただしライタ・モード時はFWE)
注2)V_{PP}:ZTAT版のみ

図6[3] RISCマイコンSH7045の内部ブロック図(144ピン版)

外部バスには図5に示すように，

- 4Mビット SRAM　TC554001AF（512Kバイト）
- 4Mビット・フラッシュ・メモリ　MBM29F040（512Kバイト）
- PWM信号発生回路（CPLD　M4A5-128/64）

が接続されます．

二足歩行ロボットの初歩的な動作には512KバイトのSRAMは必要ないかもしれませんが，将来の高機能化，高度な応用に備えることにしました．外部接続したフラッシュ・メモリはシステム・パラメータの記録やプログラムの格納に使うことができます．

SH7045F内蔵のフラッシュ・メモリは，書き換え回数が100回しか保証されていないので，デバッグ段階で何回もデータを書き換えるような使い方はできません．

外部接続のMBM29F040は10万回書き換え可能な汎用フラッシュ・メモリです．デバッグ・プログラムや調整パラメータを何回書き換えても大丈夫です．また，プログラム実行中にCPUからのアクセスでデータを書き換えることもできます．調整段階で何回も書き換えが必要なロボットの各関節のオフセット値やモーション・データの記憶に使うことができます．

図7はROBO-01Aのメモリ・マップです．SH7045Fの外部アドレス空間はCS0～CS3に分割されていますが図のように

CS0：内蔵フラッシュ・メモリが一部占有．外部では使わない
CS1：4Mビット SRAM（TC554001AF）
CS2：4Mビット・フラッシュ・メモリ（MBM29F040C）
CS3：PWM制御レジスタ（PWM0～PWM35）

と配置しました．

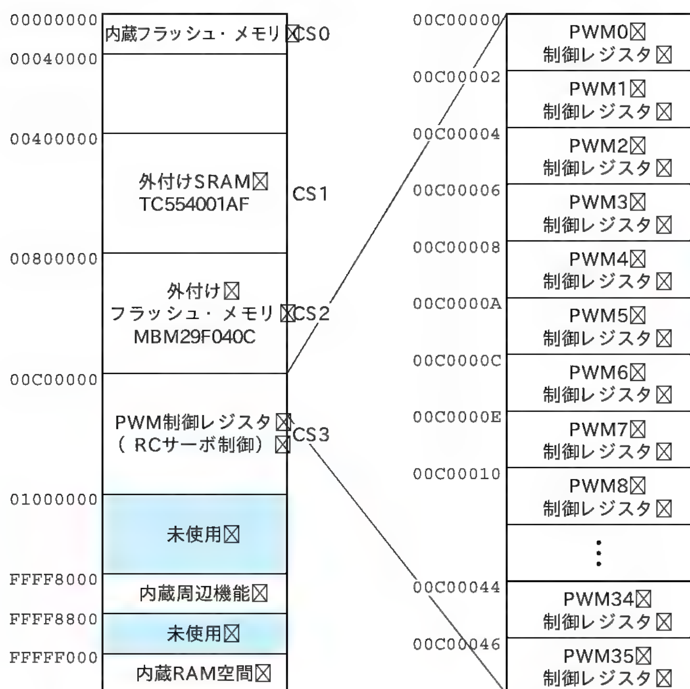


図7　二足歩行ロボット制御基板ROBO-01Aのメモリ・マップとPWM制御レジスタの配置（SH7045F）

メモリ空間CS2に配置するMBM29F040CはCPUのメイン・メモリに使える高速フラッシュ・メモリです．データの消去および書き込みも特別なプログラミング装置や高い電圧は必要としません．基板に実装した状態でCPUのプログラムにより消去と書き込みが可能です．

この基板の最大の特徴は，RCサーボ・モータの制御回路にあります．アドレス空間00C00000H～00C00046Hに配置された10ビットのPWM制御レジスタに角度情報値（0～1023）を書き込むと，CPLDで構成されたハードウェアがPWM信号を発生します．

リスト1は，アドレス00C00000Hに配置されたPWM制御レジスタPWM00に制御角度データ_angle_dataをセットするプログラムです．メモリにワード・データを書き込むのとまったく同じ方法で，36個のPWM制御レジスタを書き換えることができます．

表5の仕様書に示すようにROBO-01Aは36チャネル（＝自由度）のPWM制御信号を発生させることができます．プログラマは各チャネルごとに用意された10ビットのPWMレジスタに角度情報を書き込むだけです．あとはハードウェアが定められたインターバルでPWM信号を発生させます．

SH7045F内蔵タイマを駆使する方式と比べて，CPUの負担を大幅に減らすことができます．ちょっと複雑なマルチファンクション・タイマ・ユニットを操作する必要もありません．処理にゆとりが生まれ，CPUパワーを高度な制御に当てることができます．

ラティスセミコンダクター社のCPLD　M4A5-128/64はISP（イン・システム・プログラミング）対応デバイスです．パソコン上で開発したロジック・データはJTAGケーブルを介して基板上の6個のデバイスに一括して書き込むことができます．JTAGケーブル以外，特別なプログラミング装置は必要ありません．一度書き込まれたロジック・データはチップ内蔵のEEPROMに記憶されるので電源を切っても消えることはありません．



二足歩行ロボット制御基板の製作

ロジック開発およびダウンロード手順は次章で詳しく紹介します．図8（pp.70-71）は二足歩行ロボット制御基板ROBO-01Aの回路です．M4A5のJTAG端子は10ピンJTAGダウンロード・コネクタ，J4にカスケード接続されています．

リスト1　PWM制御レジスタPWM00に制御角度パラメータを書き込む

```
#define PWM00   (*((unsigned short *) 0x00C00000))
unsigned int _angle_data;

PWM00 = _angle_data;
```

表6　二足歩行ロボット制御基板ROBO-01Aのコネクタ・ピン配列

コネクタ　J7

ピン	信号名	ピン	信号名	ピン	信号名
A1	PWM0	B1	+V(電源)	C1	GND
A2	PWM1	B2	+V(電源)	C2	GND
A3	PWM2	B3	+V(電源)	C3	GND
A4	PWM3	B4	+V(電源)	C4	GND
A5	PWM4	B5	+V(電源)	C5	GND
A6	PWM5	B6	+V(電源)	C6	GND
A7	PWM6	B7	+V(電源)	C7	GND
A8	PWM7	B8	+V(電源)	C8	GND

コネクタ　J8

ピン	信号名	ピン	信号名	ピン	信号名
A1	PWM8	B1	+V(電源)	C1	GND
A2	PWM9	B2	+V(電源)	C2	GND
A3	PWM10	B3	+V(電源)	C3	GND
A4	PWM11	B4	+V(電源)	C4	GND
A5	PWM12	B5	+V(電源)	C5	GND
A6	PWM13	B6	+V(電源)	C6	GND
A7	PWM14	B7	+V(電源)	C7	GND
A8	PWM15	B8	+V(電源)	C8	GND

コネクタ　J9

ピン	信号名	ピン	信号名	ピン	信号名
A1	PWM16	B1	+V(電源)	C1	GND
A2	PWM17	B2	+V(電源)	C2	GND
A3	PWM18	B3	+V(電源)	C3	GND
A4	PWM19	B4	+V(電源)	C4	GND
A5	PWM20	B5	+V(電源)	C5	GND
A6	PWM21	B6	+V(電源)	C6	GND
A7	PWM22	B7	+V(電源)	C7	GND
A8	PWM23	B8	+V(電源)	C8	GND

1) PWM0～PWM35はRCサーボ制御用PWM信号出力(CPLD　M4A5128/64によるハードウェアPWM).
2) *印のPWM24～PWM35が実装されているのはROBO_01Aだけ.
3) TIOC0A～TIOC0BはSH7045Fのカウンタ出力．ソフトウェアPWM信号出力として利用可能.
4) PWM0～PWM35およびTIOC0A～TIOC0Dはバス・ドライバIC 74HCT541により駆動されている.
5) コネクタ J7～J11の+V(電源)は基板の供給電圧．RCサーボの電源を想定している.

コネクタ　J10

ピン	信号名	ピン	信号名	ピン	信号名
A1	*PWM24	B1	+V(電源)	C1	GND
A2	*PWM25	B2	+V(電源)	C2	GND
A3	*PWM26	B3	+V(電源)	C3	GND
A4	*PWM27	B4	+V(電源)	C4	GND
A5	*PWM28	B5	+V(電源)	C5	GND
A6	*PWM29	B6	+V(電源)	C6	GND
A7	*PWM30	B7	+V(電源)	C7	GND
A8	*PWM31	B8	+V(電源)	C8	GND

コネクタ　J11

ピン	信号名	ピン	信号名	ピン	信号名
A1	*PWM32	B1	+V(電源)	C1	GND
A2	*PWM33	B2	+V(電源)	C2	GND
A3	*PWM34	B3	+V(電源)	C3	GND
A4	*PWM35	B4	+V(電源)	C4	GND
A5	TIOC0A	B5	+V(電源)	C5	GND
A6	TIOC0B	B6	+V(電源)	C6	GND
A7	TIOC0C	B7	+V(電源)	C7	GND
A8	TIOC0D	B8	+V(電源)	C8	GND

コネクタ　J15

ピン	信号名	ピン	信号名	ピン	信号名
A1	TIOC4A	B1	+ V_{CC}	C1	GND
A2	TIOC4B	B2	+ V_{CC}	C2	GND
A3	TIOC4C	B3	+ V_{CC}	C3	GND
A4	TIOC4D	B4	+ V_{CC}	C4	GND
A5	PB2	B5	+ V_{CC}	C5	GND
A6	PB3	B6	+ V_{CC}	C6	GND
A7	PB4	B7	+ V_{CC}	C7	GND
A8	PB5	B8	+ V_{CC}	C8	GND

コネクタ　J14

ピン	信号名	ピン	信号名	ピン	信号名
A1	TIOC1A	B1	+ V_{CC}	C1	GND
A2	TIOC1B	B2	+ V_{CC}	C2	GND
A3	TIOC2A	B3	+ V_{CC}	C3	GND
A4	TIOC2B	B4	+ V_{CC}	C4	GND
A5	TIOC3A	B5	+ V_{CC}	C5	GND
A6	TIOC3B	B6	+ V_{CC}	C6	GND
A7	TIOC3C	B7	+ V_{CC}	C7	GND
A8	TIOC3D	B8	+ V_{CC}	C8	GND

コネクタ　J13

ピン	信号名	ピン	信号名	ピン	信号名
A1	PD24	B1	+ V_{CC}	C1	GND
A2	PD25	B2	+ V_{CC}	C2	GND
A3	PD26	B3	+ V_{CC}	C3	GND
A4	PD27	B4	+ V_{CC}	C4	GND
A5	PD28	B5	+ V_{CC}	C5	GND
A6	PD29	B6	+ V_{CC}	C6	GND
A7	PD30	B7	+ V_{CC}	C7	GND
A8	PD31	B8	+ V_{CC}	C8	GND

コネクタ　J15

ピン	信号名	ピン	信号名	ピン	信号名
A1	PD16	B1	+ V_{CC}	C1	GND
A2	PD17	B2	+ V_{CC}	C2	GND
A3	PD18	B3	+ V_{CC}	C3	GND
A4	PD19	B4	+ V_{CC}	C4	GND
A5	PD20	B5	+ V_{CC}	C5	GND
A6	PD21	B6	+ V_{CC}	C6	GND
A7	PD22	B7	+ V_{CC}	C7	GND
A8	PD23	B8	+ V_{CC}	C8	GND

コネクタ　J1

ピン	信号名	ピン	信号名	ピン	信号名
A1	AN0	B1	+ V_{CC}	C1	GND
A2	AN1	B2	+ V_{CC}	C2	GND
A3	AN2	B3	+ V_{CC}	C3	GND
A4	AN3	B4	+ V_{CC}	C4	GND
A5	AN4	B5	+ V_{CC}	C5	GND
A6	AN5	B6	+ V_{CC}	C6	GND
A7	AN6	B7	+ V_{CC}	C7	GND
A8	AN7	B8	+ V_{CC}	C8	GND

6) TIOC1A～TIOC4DはSH7045Fのカウンタ出力．ソフトウェアPWM信号出力として利用可能.
7) PB2～PB5およびPD16～PD31はSH7045Fのパラレル・ポート入出力ピン.
8) AN0～AN7はSH7045FのA-Dコンバータ入力ピン．センサのアナログ入力などに使用可能.
9) コネクタ J7～J11の+ V_{CC} は+5V．センサなどの電源として使うことができる.

コネクタ J6(Bluetoothなどを接続)

ピン	信号名	ピン	信号名
1	+VCC	2	+VCC
3	RTS	4	CTS
5	RXD	6	TXD
7	GND	8	GND

コネクタ J2(ダウンロード・ケーブル接続端子)

ピン	信号名
1	GND
2	TXD1
3	RXD1

コネクタ P1

ピン	信号名	ピン	信号名
1	(DCD)	6	(DTR)
2	TXD0	7	(CTS)
3	RXD0	8	(RTS)
4	(DSR)	9	NC
5	GND		

コネクタ J2(ダウンロード・ケーブル接続端子)

ピン	信号名	ピン	信号名
1	モータ1+	2	モータ1−
3	モータ2+	4	モータ2−
5	モータ3+	6	モータ3−
7	モータ4+	8	モータ4−
9	M_V+	10	M_GND

コネクタ JP3(基板電源)

ピン	信号名	ピン	信号名
1	+V(電源)	2	GND

コネクタ J3(基板電源)

ピン	信号名	ピン	信号名
1	+V(電源)	2	GND

10) プログラムをダウンロードする時は，付属のケーブルによりコネクタJ2とパソコンのシリアル・ポートを接続する.
11) コネクタJ6はロジック電圧レベルのシリアル信号端子．Bluetoothモジュールなどを接続する.
12) コネクタP1はRS-232Cシリアル信号端子．パソコンCOMポートとストレート結線ケーブルで接続する.
13) コネクタJP2はDCモータのブリッジ駆動回路．端子(M_V+)(GND)間にDCモータの電源を接続する．基板供給電源(+V)を使う場合は，はんだランド・ジャンパJ5をショートする.
14) コネクタJP3およびJ3は基板供給電源端子．RCサーボに対応した電圧(+5V，+6V，+7.2Vなど)を供給する.

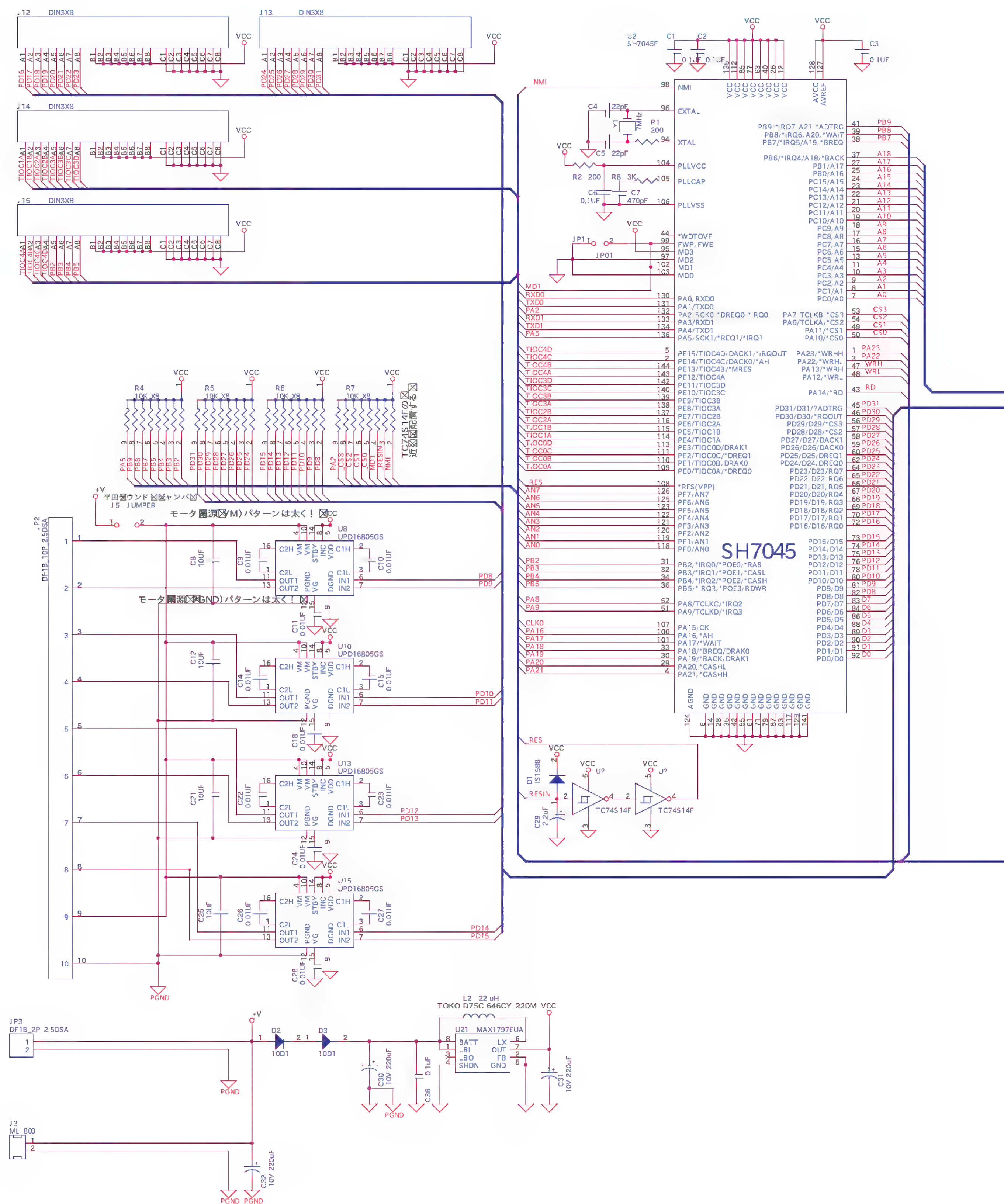


図8 二足歩行ロボット 制御基板 ROBO-01A の回路図



図8　二足歩行ロボット 制御基板 ROBO-01A の回路図（つづき）

Information ―― Xilinx，CMC と提携してインドに FPGA の技術開発センタを設立
米 Xilinx 社は，インドの Computer Maintenance 社（CMC）と提携して，インドのハイドラバードに FPGA の技術開発センタ「Xilinx-CMC India Development Center」を開設すると発表した．CMC は，リアルタイム・ハードウェアおよびソフトウェアのベンダである．

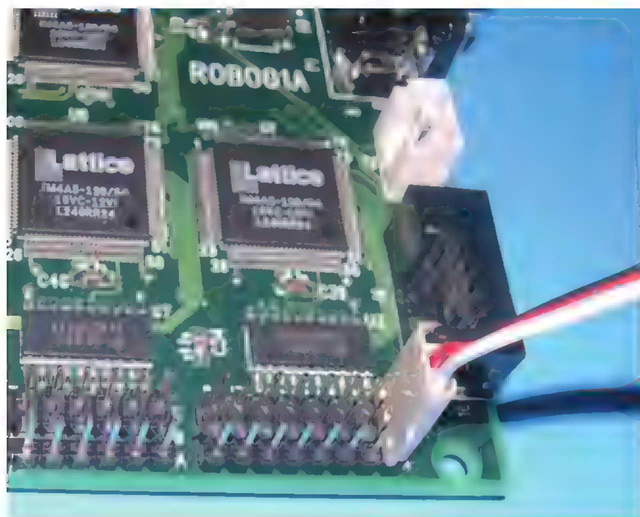


写真３　サーボ・モータをワンタッチで接続できる３列コネクタ

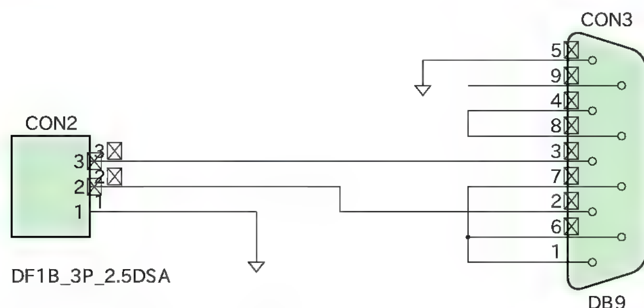


図９　ダウンロード・ケーブルの結線図



写真４　小型ギアド・モータ



写真５　３芯のダウンロード・ケーブル

CPLD の PWM 出力信号は直接コネクタに出力せず，バス・ドライバ 74HCT541 を介して出力しています．サーボ・モータの結線ミスによる CPLD の破損を避けるためです．20ピン SOP の 74HCT541 であれば交換も容易でしょう．

PWM 信号は表６（p.69）に示すように，３列×８ピンのコネクタ J7，J8，J9，J10，J11 に出力しています．サーボ・モータの３ピン・コネクタは写真３に示すようにワンタッチで接続できます．信号配列は近藤科学，双葉電子のサーボ・モータに合わせました．

二足歩行ロボット制御基板 ROBO-01A のコネクタ J11，J14，J15 にはタイマ信号も出力されています．初代 FREEDUM で採用された内蔵タイマ方式による PWM 信号出力もできます．PWM 出力として使わない場合は，パラレル入出力ポートとして利用できます．

表６のコネクタはすべて３列×８ピンのコネクタを使いました．センサ入力やディジタル入出力の配線において，電源線と信号線をペアにしておくと便利なことが多いからです．

ROBO-01A は DC モータ・ブリッジ駆動回路も４軸備えています．駆動部によっては RC（ラジコン）サーボ・モータよりギアド・モータのほうが便利なこともあります．最近は写真４のような小型で高性能なギアド・モータも容易に入手できるようになりました．

SH7045F は２チャネルのシリアル通信インターフェースを内蔵しています．このうちチャネル１は汎用の RS-232-C ポートとして９ピン D-SUB コネクタ（メス）に出力しています．

このコネクタはパソコンの COM ポートとストレート結線のシリアル・ケーブル（オス・メス）で接続でき，リモコン用の無線 LAN モジュールなどを接続します．

チャネル１（TxD0，RxD0）の TTL レベル信号はコネクタ J6 に出力しています．ロボットの通信手段として Bluetooth モジュールなど TTL レベル信号のほうが便利なことがあるからです．９ピン D-SUB の信号とは競合しますが，TTL レベルのシリアル・デバイスを接続した場合はこちらが優先されます．

チャネル２も電圧レベルは RS-232-C 準拠です．ダウンロードおよびデバッグ用に使うことを考えて，シンプルな３線（TxD，RxD，GND）構成にしました．

図９はダウンロード・ケーブルの結線図です．ROBO-01A とパソコンの COM ポートをこのケーブルで接続し，内蔵フラッシュ・メモリへの書き込み，外部 SRAM を使ったデバッグなどを行います．

二足歩行ロボットのデバッグはメカのバランスをとりながらモーション・パラメータを決めていく作業が中心になりますが，腰の強い市販のRS-232-Cケーブルではロボットのバランスを損うので実用的ではありません．写真5のような細い3芯ケーブルで自作したダウンロード・ケーブルは，モーション・デバッグ作業の必需品です．

電源回路もひとくふうしてみました．市販のRCサーボ・モータの電源仕様は＋4.8V～＋6V程度の広い範囲にわたっています．表6のRCサーボ・モータの電源ピン〔＋V（電源）〕にはROBO-01Aの電源コネクタJ3（もしくはJP3）の外部供給電圧をそのまま出力します．

近藤科学のサーボ・モータPDS-2144FETの定格電圧は6Vですが，多くのユーザ・レポートで報告されているように，7Vくらいの過電圧を加えても破損することはなさそうです．

たとえば，筆者が入手したニカド5本組み電池5N-700AACL（公称6V/700mAh，三洋電機）は，満充電で開放端子電圧は7V近くになります．しかし，1～2A程度の負荷をとると6.3V程度まで落ち着くので，そのままROBO-01Aの供給電源として使うことにしました．

図8の回路図で＋Vと表示されている電源端子にはこの供給電圧が接続されます．表6の〔＋V（電源）〕はサーボの電源端子ですが，この電圧が供給されています．

二足歩行ロボット制御基板ROBO-01AのCPU SH7045Fをはじめロジック回路は＋5V±0.25Vの安定化電源を必要とします．図8の回路では外部供給電源をダイオードで一度5.5V以下にドロップさせます．この出力電圧を再びステップアップ・レギュレータMAX1797で昇圧して＋5Vを作ります．回路図のV_{CC}はこのレギュレータから供給される電圧です．

サーボ・モータに強いトルクが加わると大電流が流れ，電池電圧は4V近くまで下がることがありますが，このときもMAX1797の働きにより安定した＋5Vが得られます．

MAX1797はたいへん強力なステップアップ・レギュレータです．入力電圧が2.5Vくらいに低下しても＋5V，500mAの出力が可能です．サーボ電源と制御回路電源の両方を1個のニカド組み電池（5N-700）で供給していますが，このレギュレータのおかげで何のトラブルもありません．

ROBO-01Aは二足歩行ロボット制御に必要な機能をコンパクトにまとめた基板です．試作ロボットWSGH-1に組み込み，歩行アルゴリズムおよびソフト開発を経て機能の実証も完了しました．

この基板を製作してみて思わぬ発見がありました．それはROBO-01Aが二足歩行ロボットの制御だけでなく広い用途に利用可能であるということです．

アナログ/ディジタル入力とディジタル出力を備えたSH-2＋CPLDという組み合わせは，いろいろな用途に利用できます．工場内の自動制御装置とか，簡単なアクチュエータ制御装置として使いたいという話を聞きました．ここでも「二足歩行ロボット制御技術は組み込み制御技術の集大成」という事実を実感しました．



二足歩行ロボット制御基板の機能を絞った小型モデルの開発

イトーレイネツ社のブラケットServo Creation（SCT-01KO）およびオプション部品Servo Creation Option2（SCOP-002）を使って標準的なロボットを試作する限りでは前項で紹介した基板のサイズで十分でした．しかし，次のステップとして「強いロボット」を目指す段階で，基板の小型化がクローズアップされてきました．

実際にロボットを動かすまではわからなかった，細かい使い勝手もわかってきました．小型ロボットの場合は重量バランスが重要です．制御基板も本当に必要な機能を残して絞り込む必要があります．そこで本執筆と並行してロボット制御基板第2弾の開発を進めました．それが次に紹介する二足歩行ロボット制御基板ROBO-02です．

まずマイクロプロセッサですが，SH7045F（28MHz）をSH7047F（50MHz）に変更しました．目的はチップの小型化とCPUクロックの高速化です．

CPUクロックの高速化のためであればSH7145F（50MHz）という選択肢もありますが，電源電圧が3.3Vになります．これは省電力設計の観点からは好都合ですが，センサとのマッチングに問題を残します．

CPUの電圧が3.3VになるとA-Dコンバータの入力電圧も0～3.3Vに制限されてしまいます．センサの世界も少しずつ

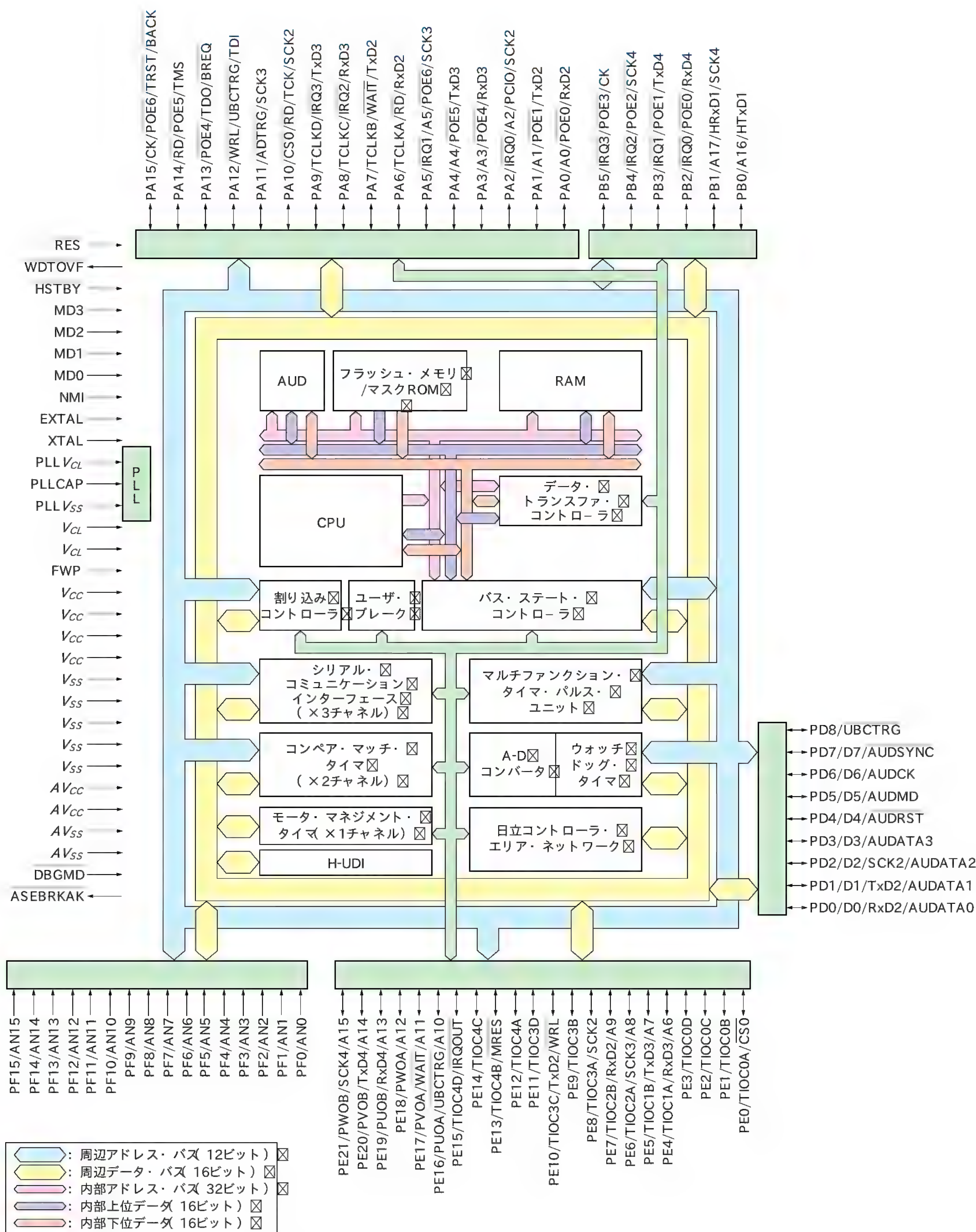


図10[4]　RISCマイコンSH7047の内部ブロック図

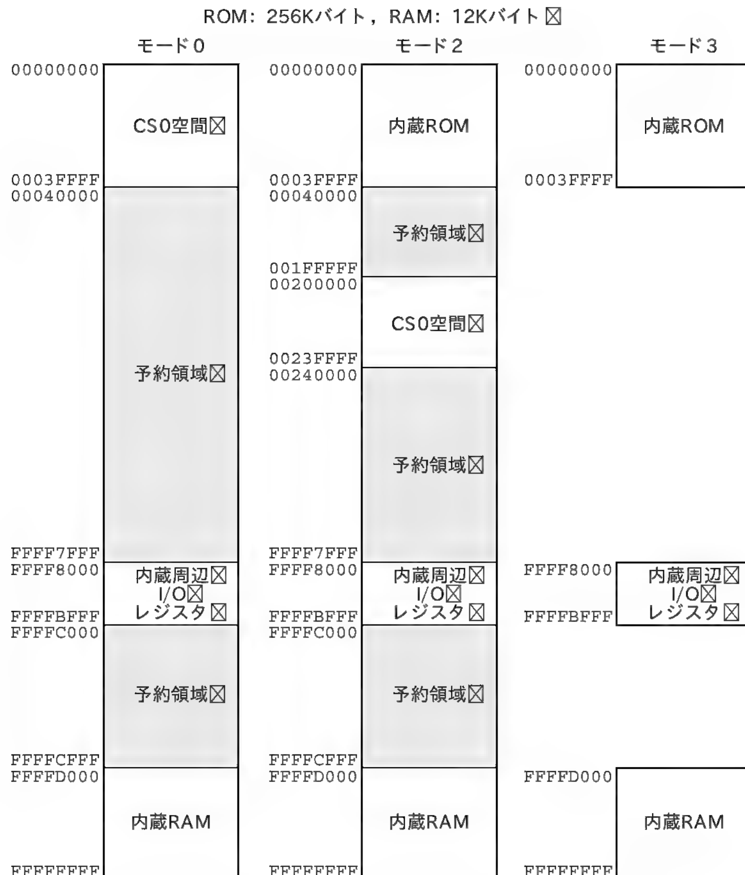


図11　SH7047フラッシュ・メモリ版の各動作モードのアドレス・マップ

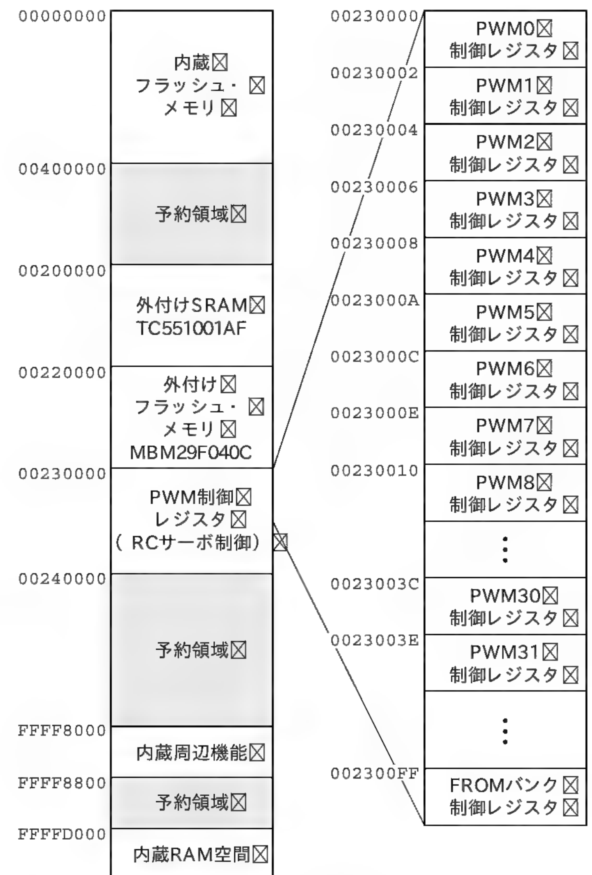


図12　二足歩行ロボット制御基板ROBO-02のメモリ・マップとPWM制御レジスタの配置（SH7047F）

3.3V化は進んでいますが，まだまだ5V出力のセンサも少なくありません．それにSH7045FとSH7145Fはピン互換でパッケージ寸法は24×24mmと同じ大きさです．

そこでパッケージ寸法16×16mmのSH7047Fを選択しました．図10はRISCマイコンSH7047Fの内部ブロック図です．

SH7047のA-Dコンバータ入力は16チャネルあります．SCI（シリアル・コミュニケーション・インターフェース）も3チャネル内蔵し，CANインターフェースも備えています．電源電圧は＋5V，クロック速度も50MHzと，いいことづくめのようですが，100ピン・パッケージにともなう制約もあります．まず外部データ・バスは8ビットに固定，外部アドレス空間は実質256Kバイトです．

図11はSH7047Fのアドレス・マップです．ROBO-02はモード2で使いますが，外部に解放されるのはCS0空間です．二足歩行ロボット制御基板ROBO-02は図12に示すように，

00200000～0021FFFF　SRAM（TC551001AF）
00220000～0022FFFF　フラッシュ・メモリ（MBM29F040C）
00230000～0023FFFF　PWM制御レジスタ

と配置します．

フラッシュ・メモリ（MBM29F040C）の容量は512Kバイトありますが，これを8バンクに分割して64Kバイトのアドレス空間，

00220000～0022FFFF

に配置します．図12のメモリ・アドレス，

0023000FF

に配置された［フラッシュ・メモリ・バンク制御レジスタ］はこのバンク制御を行います．

SH7047FはCANインターフェースを内蔵していますが，その信号線（HTxD1，HRxD1）はなんと外部アドレス線（A16，A17）とピンを共有しています．CANインターフェースを使うと外部アドレス空間は64Kバイトに制限されます．

図13（稿末）は二足歩行ロボット制御基板ROBO-02の回路構成，図14は二足歩行ロボット制御基板ROBO-02の回路図です．

参考・引用＊文献
（1）科学技術振興機構，PINO仕様書
（2）＊シマフジ電機（株），VR5500-ATOMハードウェア仕様書，2002年10月
（3）＊（株）ルネサステクノロジ，SH7045Fデータシート
（4）＊（株）ルネサステクノロジ，SH7047Fデータシート
（5）富士通（株），フラッシュメモリMBM29F040データシート

よしだ・こうさく

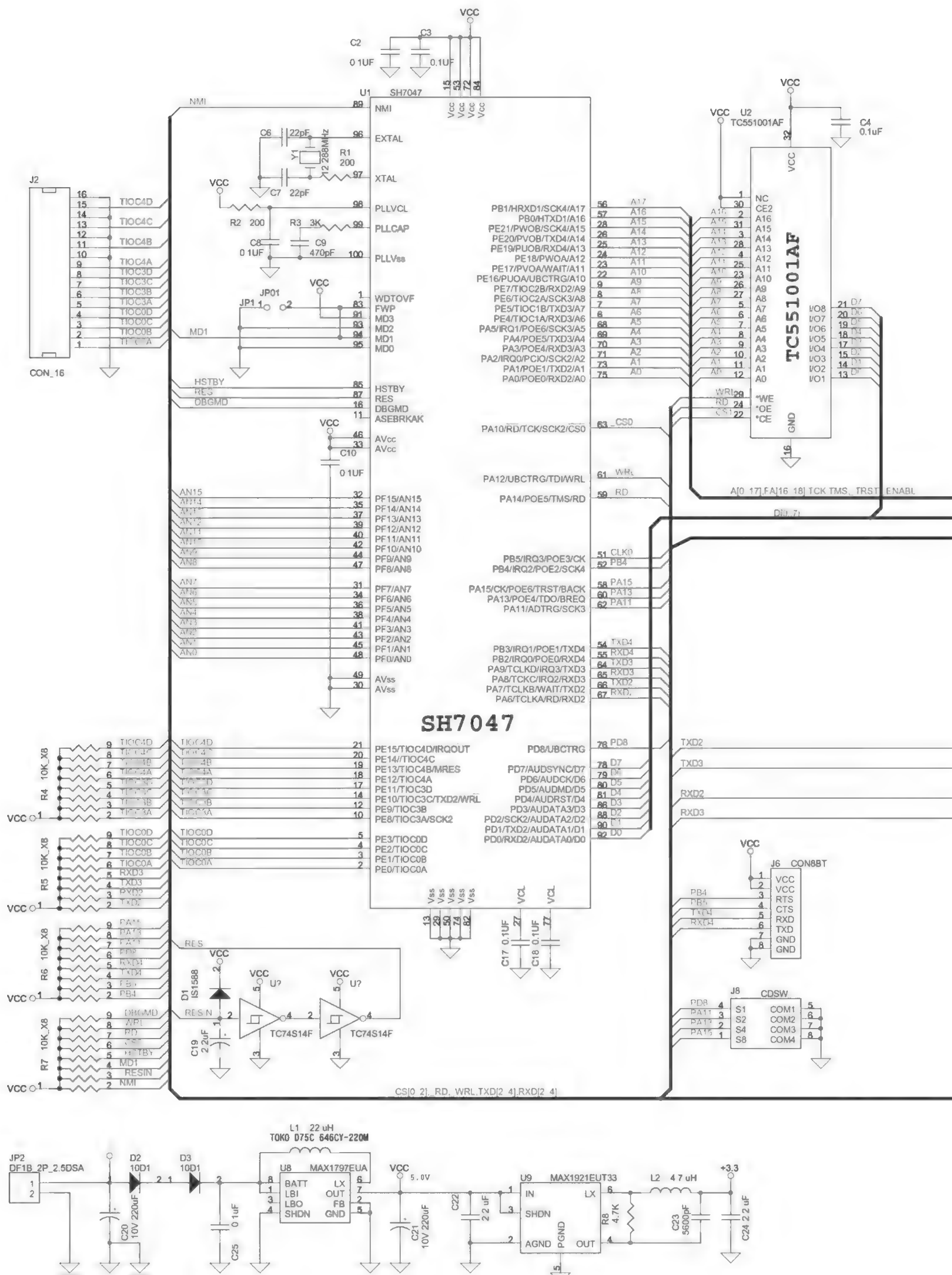


図14　二足歩行ロボット 制御基板 ROBO-02の回路図

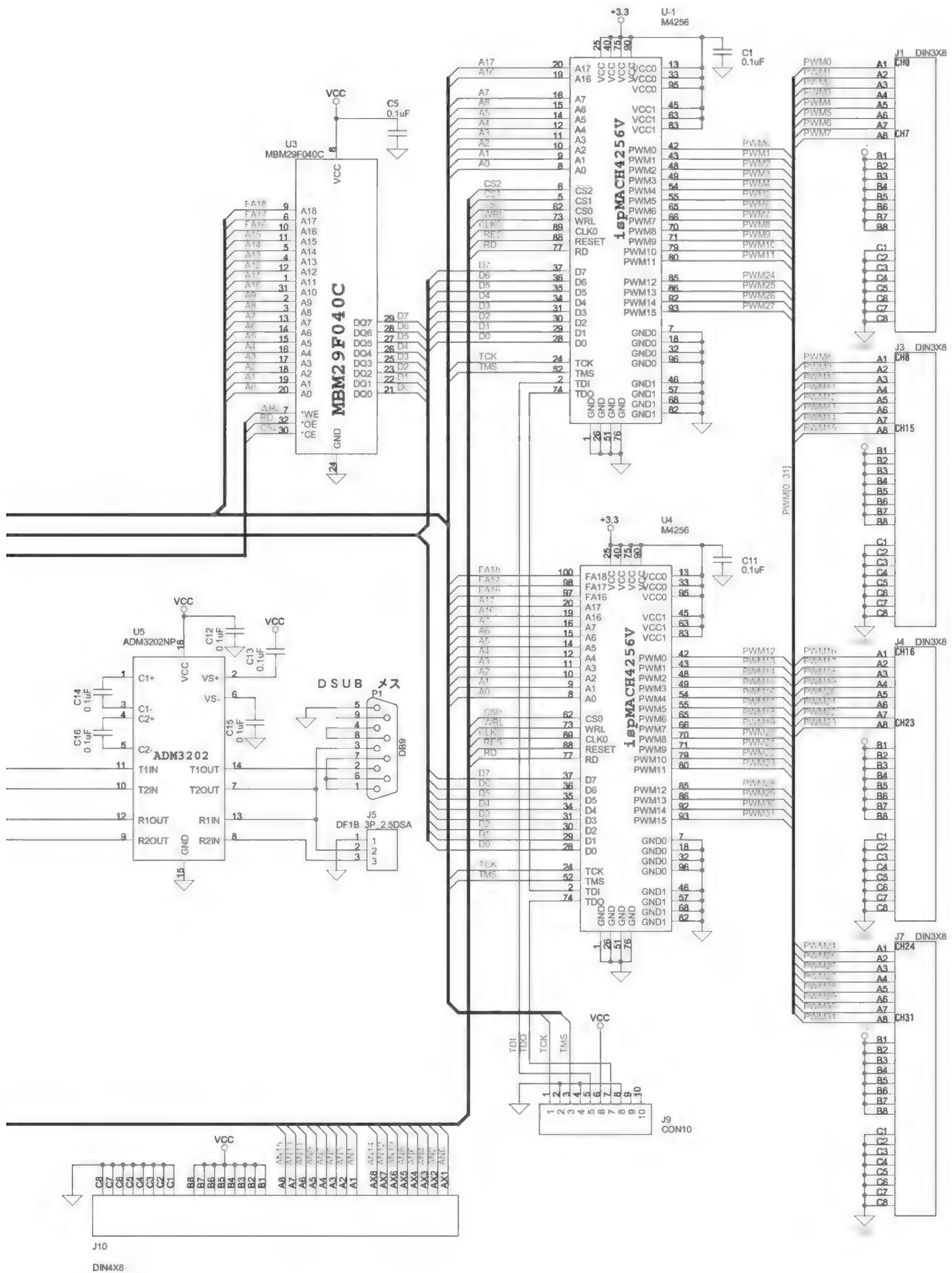


図14 二足歩行ロボット 制御基板ROBO-02の回路図(つづき)

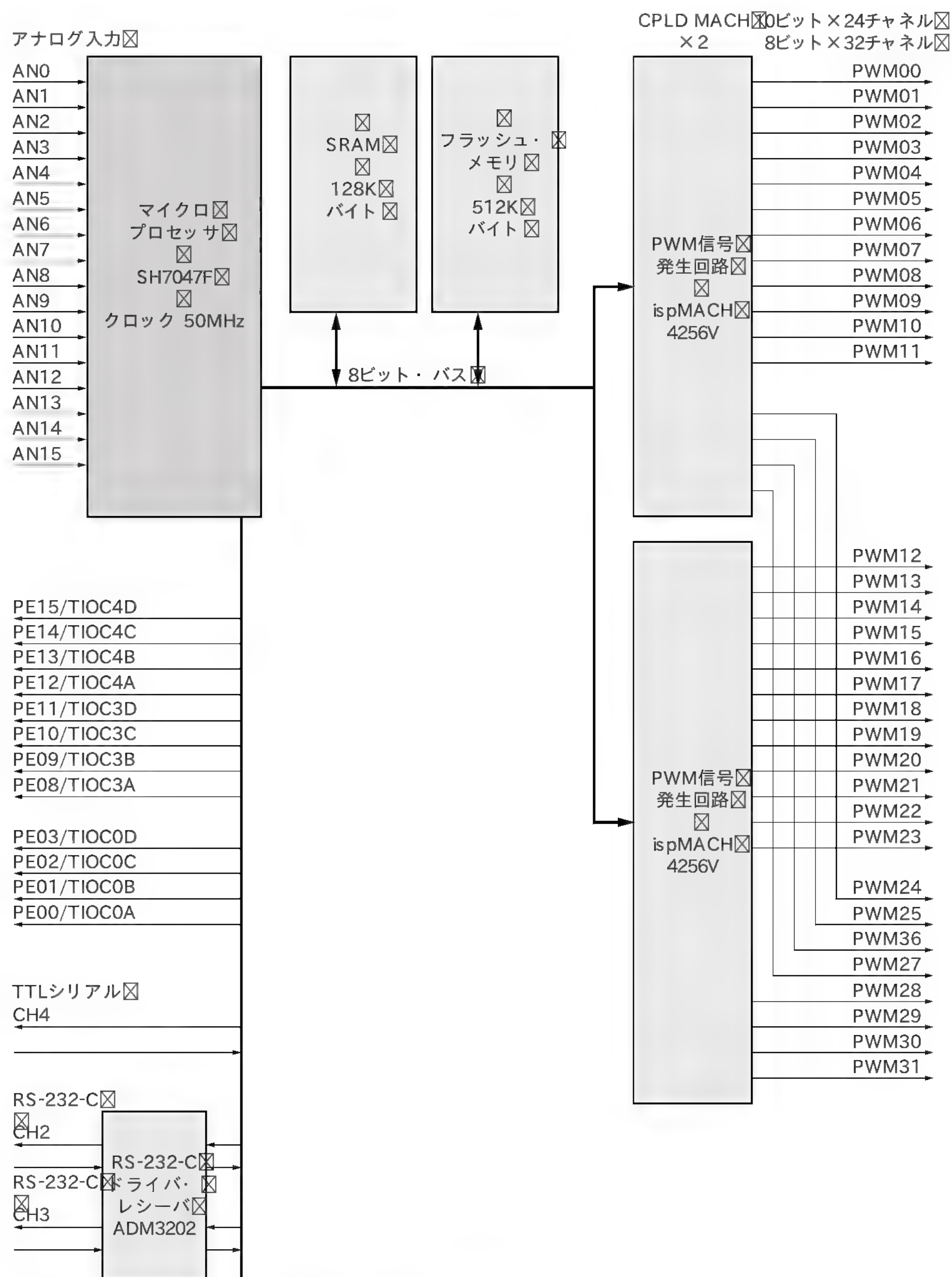
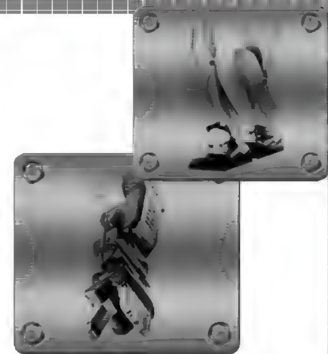


図13　二足歩行ロボット制御基板ROBO-02の回路構成

Appendix 1

フラッシュ・メモリの消去，書き込み，読み出し



吉田 幸作

MBM29F040C（富士通）は，CPUのメイン・メモリに使える高速フラッシュ・メモリです．図AはMBM29F040Cのブロック図です．読み出しは図Bに示すように通常のCPUメモリ・アクセス・サイクルで行うことができます．

データの消去および書き込みも特別なプログラミング装置や高電圧は必要としません．基板に実装した状態でCPUのプログラムにより消去と書き込みが可能ですが，少し手順を要します．

表AはMBM29F040Cのコマンド表です．チップの消去や書き込みは不用意に実行されては困りますから，ロック（錠）がかかっています．これをはずすためには決められた「実行手順」が必要です．それが表Aのコマンドです．

たとえばチップ一括消去は，

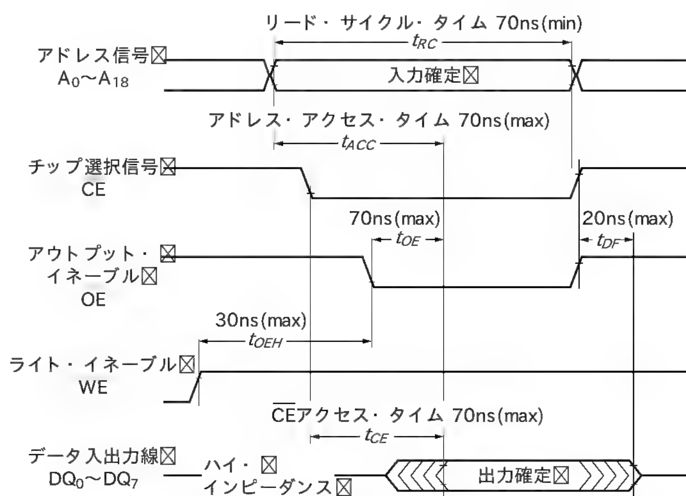
```
00555H番地 ← AAH ; 555H番地にAAHを書き込む
002AAH番地 ← 55H ; 2AAH番地に55Hを書き込む
00555H番地 ← 80H ; 555H番地に80Hを書き込む
00555H番地 ← AAH ; 555H番地にAAHを書き込む
002AAH番地 ← 55H ; 2AAH番地に55Hを書き込む
00555H番地 ← 10H ; 555H番地に10Hを書き込む
```

という一連の書き込みによりスタートします．

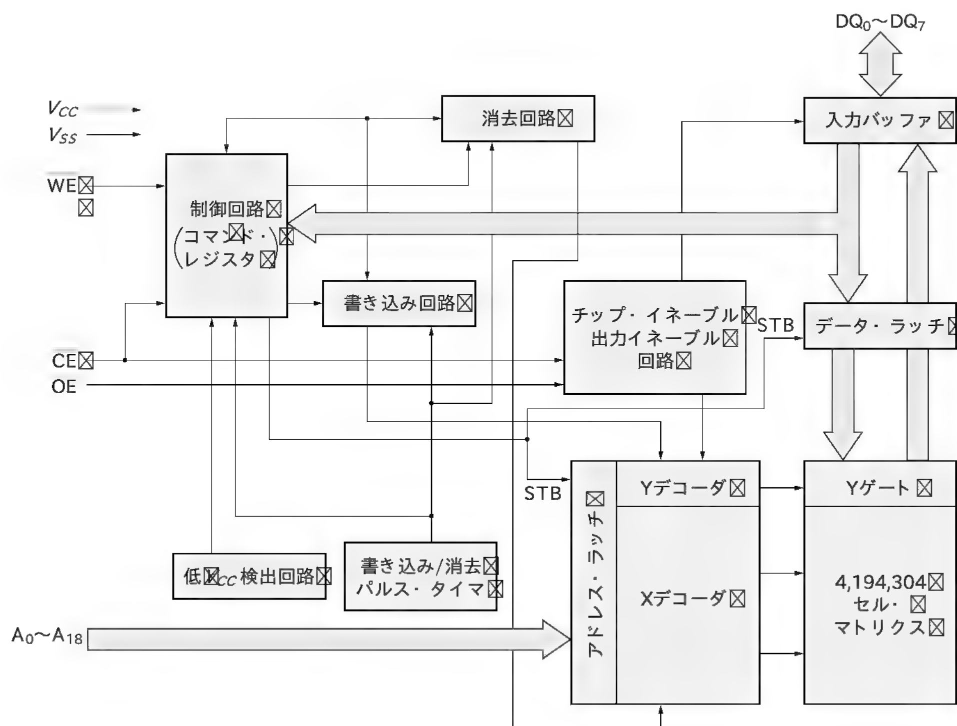
MBM29F040Cは，チップ一括消去のほかにセクタ単位の消去も可能です．セクタというのは表Bに示すように全メモリ領域を8分割した各記憶領域を指します．セクタ消去のコマンドは表Aに示すように5バイト目までは同じですが，最後に消去対象セクタ・アドレスXX000Hに，

```
XX000H番地 ← 30H ; XX000H番地に30Hを書き込む
```

点が異なります．



図B[1] 高速フラッシュ・メモリ MBM29F040C のリード・サイクル・タイミング・チャート（タイミング・パラメータはアクセス・タイム70nsのMBM29F040C-70の例）



図A[1]
高速フラッシュ・メモリ MBM29F040C の内部ブロック図

表A[1]　MBM29F040Cのコマンド

コマンド・シーケンス	バス・ライト・サイクル	1stバス・ライト・サイクル		2ndバス・ライト・サイクル		3rdバス・ライト・サイクル		4thバス・ライト/ライト・サイクル		5thバス・ライト・サイクル		6thバス・ライト・サイクル	
		アドレス	データ	アドレス	データ	アドレス	データ	アドレス	データ	アドレス	データ	アドレス	データ
リード/リセット *	1	×××H	F0H	—	—	—	—	—	—	—	—	—	—
リード/リセット *	3	555H	AAH	2AAH	55H	555H	F0H	RA	RD	—	—	—	—
エレクトロニック・シグネチャ	3	555H	AAH	2AAH	55H	555H	90H	—	—	—	—	—	—
プログラム	4	555H	AAH	2AAH	55H	555H	A0H	PA	PD	—	—	—	—
チップ・イレース	6	555H	AAH	2AAH	55H	555H	80H	555H	AAH	2AAH	55H	555H	10H
セクタ・イレース	6	555H	AAH	2AAH	55H	555H	80H	555H	AAH	2AAH	55H	SA	30H
セクタ消去一時停止	ADD(Add.="H"または"L")，Data(B0H)の入力で，セクタ消去中の消去一時停止												

RA：読み出しアドレス
PA：書き込みアドレス
SA：消去アドレス．A_{18}，A_{17}，A_{16}の組み合わせで個々のセクタを選択可能
RD：読み出しデータ
PD：書き込みデータ
×："H"または"L"
＊：2種類のリセット・コマンドは，どちらも同じ働きをする

リスト A　MBM29F040Cのセクタ＆チップ消去関数

```
#define K5      (*((unsigned char *)0x00800555))
#define KA      (*((unsigned char *)0x008002AA))

void flash_erase(int _i)          /* _i = erase sector No. */
{
        K5 = 0xAA;
        KA = 0x55;
        K5 = 0x80;
        K5 = 0xAA;
        KA = 0x55;
        if(_i == 8){
                K5 = 0x10;
                while((K5 & 0X80) == 0x00)
                        ;
        }
        else{
                _i = ( _i<<16 ) + 0x00800000;
                *((unsigned char *) _i) = 0x30;
                while((*((unsigned char *) _i) & 0X80) == 0x00)
                        ;
        }
}
```

リスト B　MBM29F040Cのバイト書き込み関数

```
#define   K5      (*((unsigned char *)0x00800555))
#define   KA      (*((unsigned char *)0x008002AA))
unsigned int _address;
unsigned char _data;

void flash_write(_address, _data)
{
        K5 = 0xAA;
        KA = 0x55;
        K5 = 0xA0;
        *((unsigned char *) _addres) = _data;
        while((*((unsigned char *) _address) ) != _data)
                ;
}
```

表B[1]　MBM29F040Cのセクタ・アドレス

セレクタ・アドレス	A_{18}	A_{17}	A_{16}	アドレス範囲
SA0	0	0	0	00000H～0FFFFH
SA1	0	0	1	10000H～1FFFFH
SA2	0	1	0	20000H～2FFFFH
SA3	0	1	1	30000H～3FFFFH
SA4	1	0	0	40000H～4FFFFH
SA5	1	0	1	50000H～5FFFFH
SA6	1	1	0	60000H～6FFFFH

リストAはROBO-01Aのフラッシュ・メモリの消去プログラムをCの関数として記述したものです．

　引き数　　`int　　_I`

は消去セクタ番号で0～7の値をとります．8を指定するとチップ一括消去を行います．

消去には数秒を要します．消去の完了は読み出しデータ・ビットDQ7のポーリングによって行います．リストAの記述

```
while((*((unsigned char *) _i) & 0X80)== 0x00)
        ;
```

は消去完了を待つプログラムです．

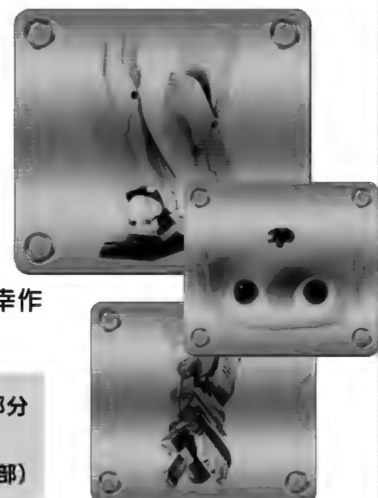
リストBは1バイト書き込み関数です．書き込みも数μs～数十μsを要するので，アドレス_addressにデータ_dataを書き込んだ後，ポーリングして書き込み完了を待ちます．

引用文献
（1）富士通，MBM29F040Cデータシート

ロボット技術者のためのロジック設計入門

3. CPLDを使用したRCサーボ信号発生回路の設計

吉田 幸作



サーボ・モータを制御する信号は，PWM信号発生回路で作る．今回はCPLDを使ってこの部分を設計する．記述言語はABELを使ったが，VHDLで書き直した例も紹介する．

（編集部）

サーボ・モータ制御用のPWM信号発生方式の比較

普及型二足歩行ロボットのアクチュエータにはRC（ラジコン）サーボ・モータが多く使われます．RCサーボ・モータの制御信号は10～20msインターバルのPWM信号なので，アクチュエータの数だけPWM信号発生回路を用意する必要があります．

図1は，よく使われるRCサーボ用のPWM信号発生方式をまとめたものです．図1（a）はいちばん簡単な，マイクロプロセッサに内蔵されているパラレル・ポートを使う方式です．PWM信号の間隔やパルス幅はマイクロプロセッサ内蔵のタイマや割り込みを使って制御します．

パラレル・ポートやタイマは，ほとんどのマイクロプロセッサに内蔵されています．また，必要なPWM信号の数だけパラレル・ポートを備えたマイクロプロセッサを使えば，特別なハードウェアを外部に用意する必要はありません．

これはコストがほとんどかからない方式ですが，欠点はすべての処理をCPUで行うため，

- CPUの処理能力の大部分がPWM信号発生処理に費やされる
- CPU処理の限界があるためPWMのビット分解能が0～180程度に制限される

など，制約が多くなることです．

図1（b）は，マイクロプロセッサ内蔵のタイマを使ってPWM信号を発生させる方式です．初代FREEDUMはこの方式を採用しています．PWM波形の発生はハードウェア（内蔵タイマ）によって行われるので，分解能はタイマのビット長まであります．

SH7045Fのタイマ（マルチファンクション・タイマ・パルス・ユニット）は16ビット分解能ですから，PWM分解能も16ビットです．この方式の欠点は，内蔵タイマのチャネル数によってPWM信号の数が制約されることです．

SH7045Fを使って32チャネルのPWM信号を発生させるにはどうしたらよいでしょうか．ひとつはマルチファンクション・タイマ・パルス・ユニットを時分割多重方式で使う方式です．図1（b）のようにタイマをワンショット・モードで時分割多重PWM波形を発生させ，これを外部のデマルチプレクサで複数チャネルのPWM信号に切り分けます．

タイマをワンショット・モードで動作させるには，毎回CPUの介入が必要になります．また，デマルチプレクサの制御もCPUの処理になるので，CPUの負担は大きくなります．

もうひとつはPWM波形発生用のマイクロプロセッサを複数

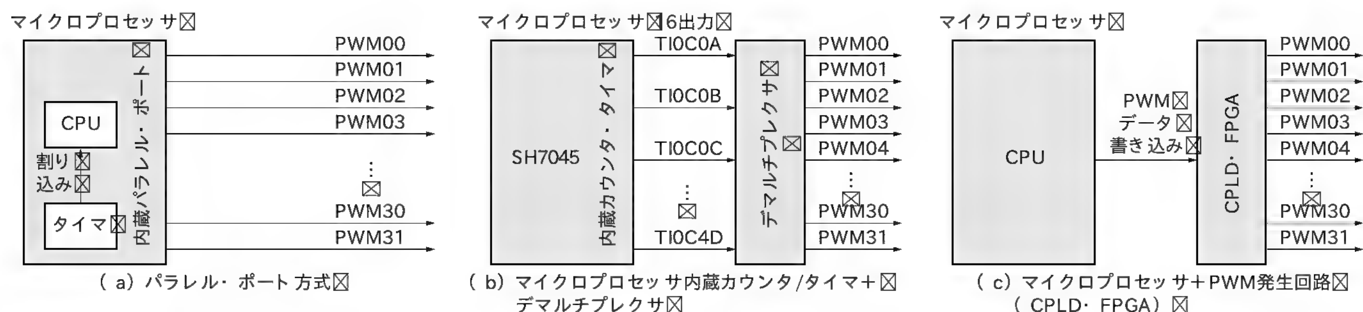


図1 RCサーボ・モータ制御のためのPWM信号発生方式

個使う方法です．12チャネルPWM信号を発生させるサブマイクロプロセッサを3個使えば合計36チャネルのサーボ・モータの制御が可能です．

図1(c)は，専用回路によってPWM信号を発生させる方式です．CPUがPWM信号発生回路のレジスタに制御データを一度書き込むだけで自動的にPWM信号を発生し続けます．

専用ハードウェアでPWM信号発生回路を作った場合のコストの検討

専用ハードウェアによるPWM信号発生回路のコストを検討してみましょう．CPLD(M4A5-128/64もしくはLC4256)の場合は24チャネルで3,000円前後です．

FPGAではどうでしょうか．第2章で紹介したROBO-02開発用に作成したABELのロジック記述ファイルをFPGA設計のプロである井倉将実氏に送り，VHDLに書き直してテストしてもらいました．Altera社のCyclone EP1C3TQ100に12チャネルのPWM信号発生回路を組み込んだときのロジック・セル使用率はなんと7%でした．

単純に計算すれば100チャネル分を組み込んでも使用率は60%程度で済み，容易に実現可能という結論が出ました．Cyclone EP1C3TQ100の小売価格は2,000円程度です．電源回路やロジック・データのロード機構に多少コストがかかっても総計3,000円のコストです．

専用ハードウェア方式は内蔵カウンタやソフトウェア・タイマを使った方法にくらべて，CPUの負荷がかなり小さくなります．また，内蔵カウンタのチャネル数によってサーボ・チャネル数の制約を受けることもありません．

二足歩行ロボットはリアルタイム性が重要です．3,000円前後のコストでシステムの自由度が大幅に向上するとなれば，これは価値のある選択です．

技術者教育におけるモチベーションの向上も二足歩行ロボットの効用の一つです．「ロボット制御のための論理設計」に挑戦してみましょう．

RCサーボ制御回路のしくみ

図2は，第2章で紹介したROBO-01Aに実装したRCサーボ制御信号発生回路のブロック図です．24～36チャネルのうちの1チャネル分を図示しました．ROBO-02の場合もクロック分周部が少し変わりますが，基本的には同じです．

まず，19ビット・フリーラン・カウンタ(Q_0～Q_{18})はCPUの基本クロック28MHzを524,288カウントします．このカウンタは各チャネル共通でCPLD1個につき1組実装します．

図3はPWM信号発生回路のタイミング・チャートですが，フリーラン・カウンタのインターバルは，

$$524{,}288/28{,}000 = 18.725\text{ms}$$

です．これがPWM信号の繰り返しインターバルになります．

サーボ信号として重要なのはPWM信号波形の幅です．繰り返しインターバルは自由度があるので，単純な2進カウンタになるようにインターバルを決めました．

図2のPWMレジスタ(PW_{00}～PW_{09})は，PWM波形の幅を決めるレジスタで10ビットの有効長をもっています．CPUはこのレジスタに8ビット・データ・バス(D_0～D_7)からPWMデータを書き込みます．

PWM信号発生回路のフリップフロップPWM0は，



図2 PWM信号発生回路のブロック図(ROBO-01A)

●フリーラン・カウンタFCNTの値が12800のときセット
●フリーラン・カウンタFCNTのQ_5〜Q_{15}とPWMレジスタPW_{00}〜PW_{10}が一致したときリセット

します．

実際に存在するPWMレジスタは10ビットですが，最上位に定数1のPW_{10}ビットを追加したPWMレジスタとフリーラン・カウンタの11ビットを比較します．

図3のタイミング・チャートに示すように，

PWMレジスタ＝0のとき	波形幅＝0.713ms
PWMレジスタ＝511のとき	波形幅＝1.298ms
PWMレジスタ＝1023のとき	波形幅＝1.883ms

となります．

RCサーボ・モータの制御特性はメーカや機種によって異なりますが，この値は試作に使ったPDS2144FETの特性に合わせました．

制御基板ROBO-01Aを使った実験では，PS401(近藤科学)やS3103(双葉電子工業)ではスパンが少し不足します．PWM波形の可変幅を大きくする必要がありますが，入力クロック周波数(28MHz)を低くするか，プリスケーラ(1/32分周カウンタ)の分周比を上げることで対応します．

図2の回路を実装するために必要なCPLDの規模を概算してみました．6チャネル分の回路を作るために必要なレジスタの数はざっと見積もって，

PWMレジスタ	10×6＝60個
PWM出力フリップフロップ	×8＝6個
フリーラン・カウンタ	19個
比較回路(組み合わせ論理)	6個
アドレス・デコーダ(組み合わせ論理)	12個

なので，最低でも合計103個の論理エレメント(レジスタ，フリップフロップ，組み合わせ論理出力)が必要です．

CPLDは，レジスタをマクロセルと呼ぶロジック基礎単位で機能を実現します．図4はラティスセミコンダクター社のispMACH4000シリーズのマクロセルです．レジスタ，フリップフロップ，組み合わせ論理出力を作る論理回路モジュールになっています．

ROBO-01Aに使ったM4A5-128/64はマクロセルを128個内蔵しています．このデバイスに103個の論理エレメントを組み込むと，マクロセル利用率は，

103/128＝80.5％

になります．

CPLDの構造については後半で少し詳しく紹介しますが，この数値はかなり高い値です．このデバイスに8組のPWMコントローラを実装しようとした試みは無謀でした

プログラマブル・デバイスCPLDの構造

CPLD(Complex Programmable Logic Device)は，図5に示すように複数のロジック生成ブロック(GLB：Generic Logic

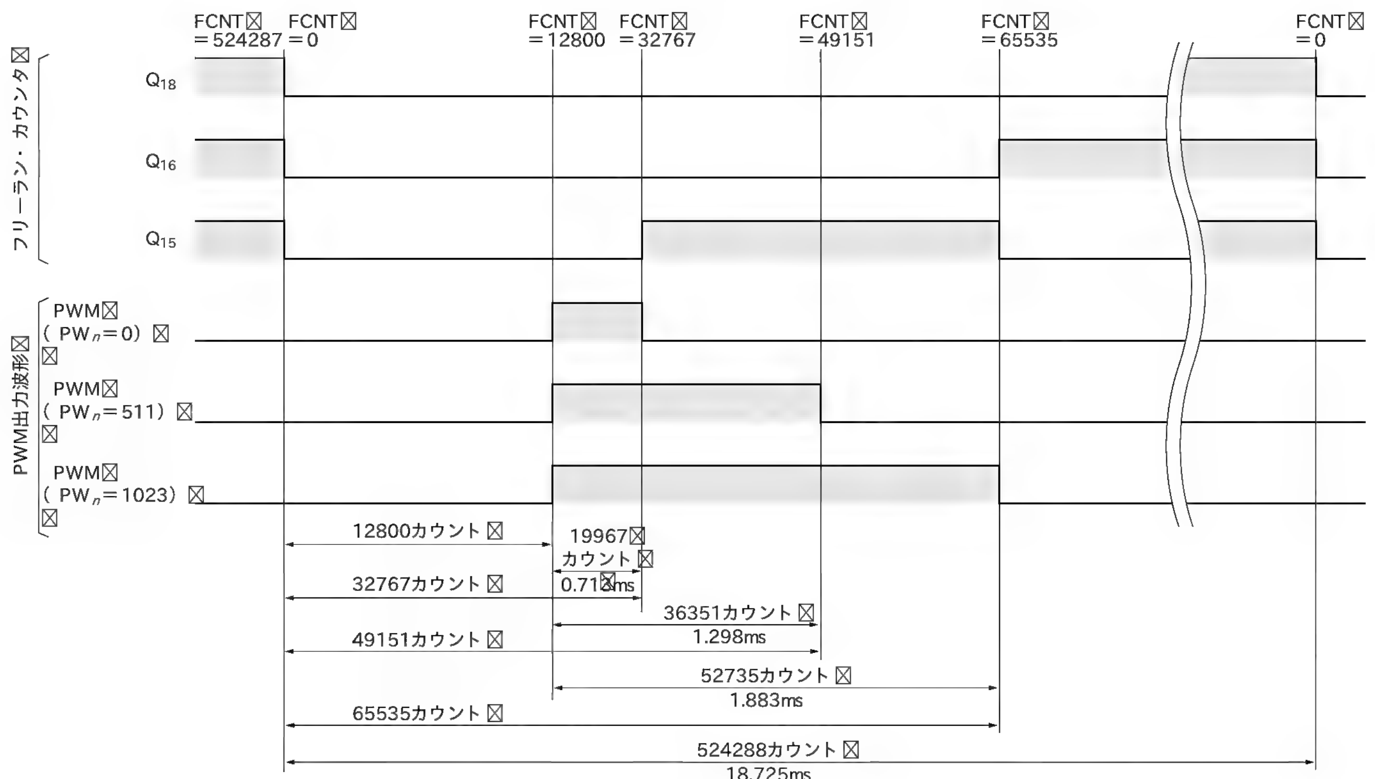


図3　サーボ制御用PWM信号発生回路のタイミング・チャート

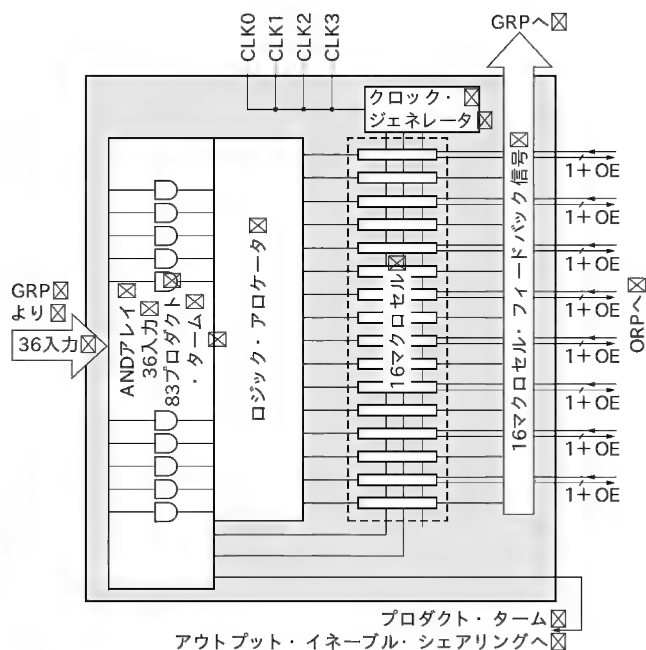


図4[1]　ispMACH4000シリーズのロジック生成ブロック

図5[1]　ispMACH4256のブロック図

Block)とブロック相互の配線領域を1チップ化したデバイスです．ロジック生成ブロックはかつてPAL，GALと呼ばれたPLD(Programmable Logic Device)1個に相当します．

図4はロジック生成ブロックの構造，図5はROBO-02に使ったispMACH4256の内部ブロック図です．1個のロジック生成ブロックには図6に示すマクロセルが16個含まれています．

マクロセルはプログラミングにより，

- 組み合わせ論理出力
- Dフリップフロップ
- T(トグル)フリップフロップ
- ラッチ

のいずれかになります．

マクロセルの入力段には，

- 36入力83積項出力ANDアレイ(図7)
- ロジック・アロケータ(図8)

が配置されています．1個のロジック生成ブロックには図7に示すANDアレイが80+3個組み込まれています．

このうち3個のANDアレイはクロック，初期化信号，OE(Output Enable)信号に使われます．残りの80個のANDアレイは，図8の個別積項アロケータにより5個ずつ論理和(OR)がとられます．

個別積項アロケータはロジック生成ブロックあたり16組用意されていますが，マクロセルと1対1に対応しているわけではありません．図8のクラスタ・アロケータにより隣接するマクロセル間で再配置が行われます．

ロジック・アロケータにより1個のマクロセルは最高25個のANDアレイの論理和を入力とすることができます．このとき

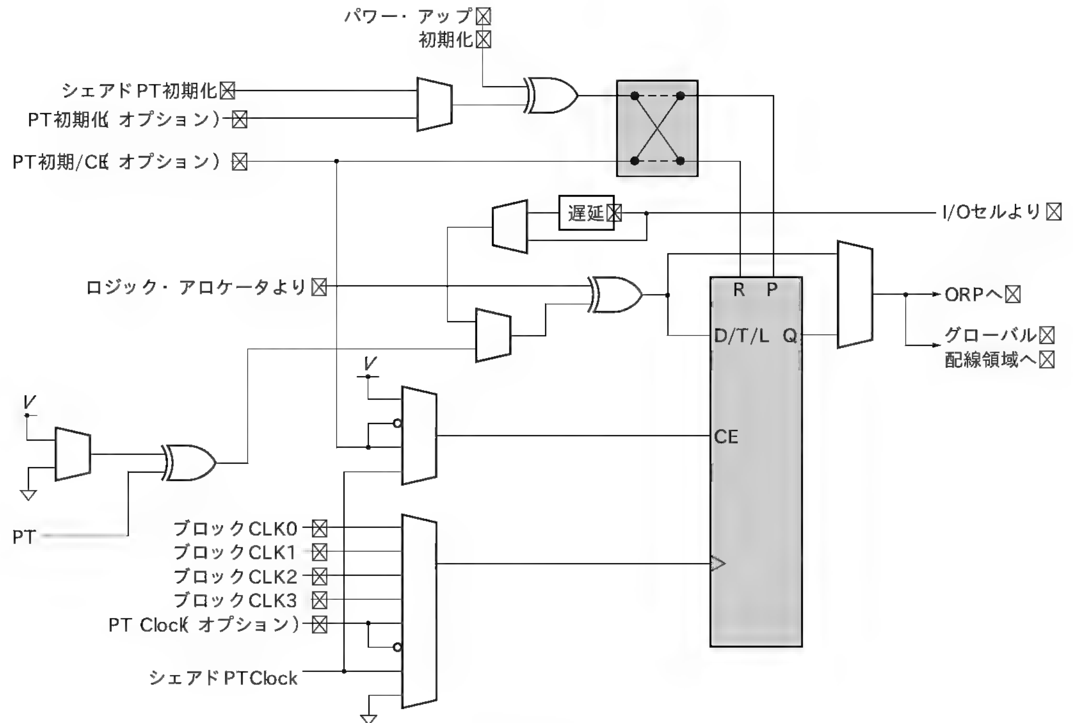
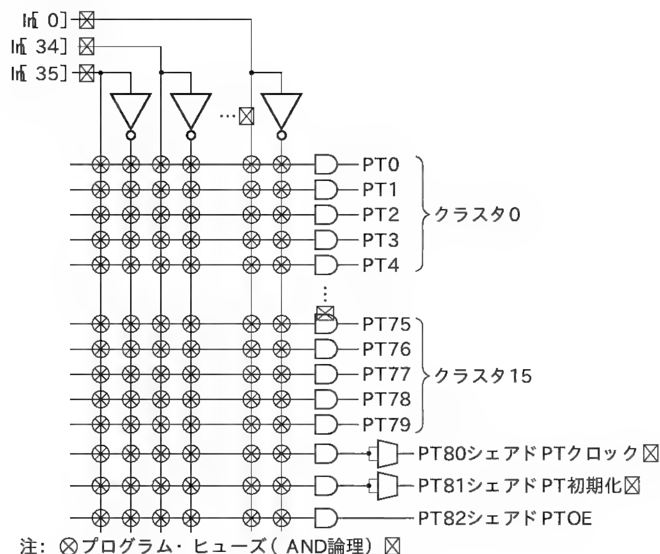


図6[1]
ispMACH4000シリーズの
マクロセル



注: ⊗プログラム・ヒューズ(AND論理)

図7[1] ispMACH4000シリーズのANDアレイ

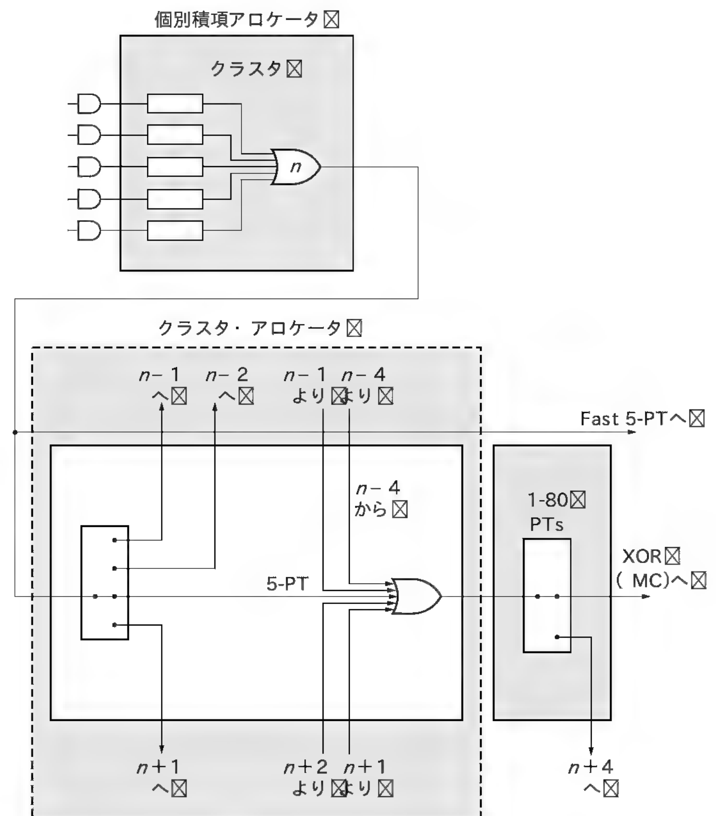


図8[1] ispMACH4000シリーズのロジック・アロケータ

ロジック生成ブロックの中には論理積和入力がまわってこないマクロセルができます．

通常，CPLDのマクロセルの割り当てとピン配置は，フィッタと呼ばれる開発ツールが自動的に行います．

ispMACH4256には図5に示すように16個のロジック生成ブロックが内蔵されています．各ロジック生成ブロックには16マクロセルが入っているので，合計で，

16× 16＝ 256個

のマクロセルが内蔵されていることになります．しかし，ロジック・アロケータの働きにより使用されないマクロセルが発生するので，使用効率が100％になることはまずありません．

マクロセル出力はORP(Output Routing Pool)と呼ばれる出力配線領域を経て図9の入出力セルに出力されます．入出力セルはピンの入出力，3ステート制御などを行うブロックです．

図5のグローバル配線領域は，ロジック生成ブロック相互の配線領域です．ispMACH4000シリーズのロジック生成ブロック入力は最高36入力です．

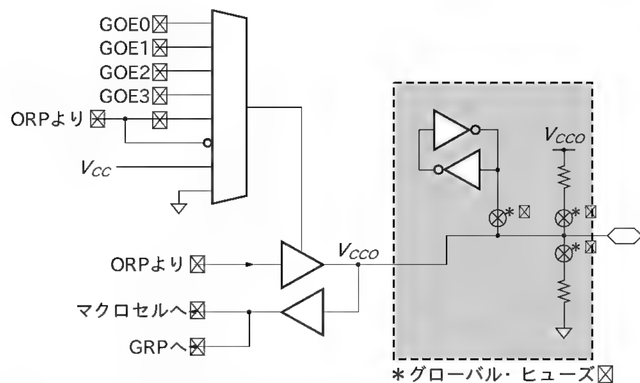


図9[1] ispMACH4000シリーズの入出力セル

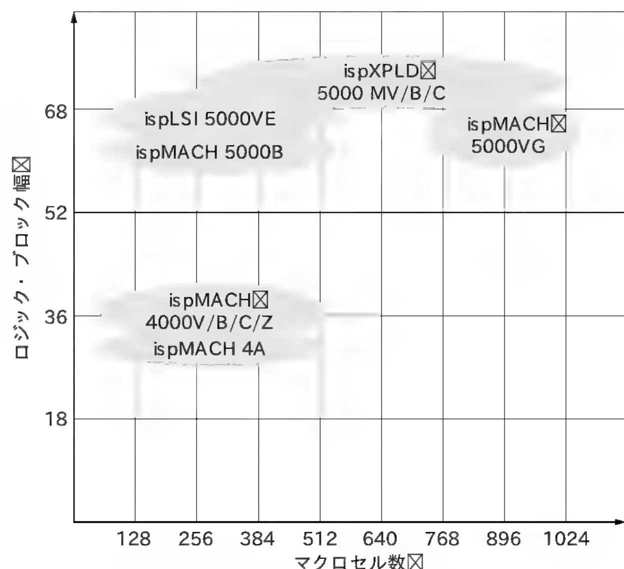


図10[1] ラティスセミコンダクター社のCPLD＆PLD製品ラインナップ



プログラマブル・デバイスCPLDの選択

● ラティスセミコンダクター社のCPLD製品

図10はラティスセミコンダクター社のCPLD製品を図示したものですが，

- ispMACH4A
- ispMACH4000シリーズ
- ispMACH5000シリーズ
- ispXPLD5000シリーズ

に大別できます．

表1は，各シリーズをコア電圧ごとに分類し特性を列挙したものです．各シリーズには電源電圧，集積度(マクロセル数)およびパッケージの異なる多くのデバイスがラインナップされています．

ispM4Aシリーズはいちばん古くからのシリーズです．ROBO-01AにはこのシリーズのispM4A5-128/64を使いました．電源電圧5Vの製品はこのispM4A5だけです．電源電圧は5Vですが出力電圧は最高＋3.3Vに制限されているので，5V/3.3V混在回路ではレベル・コンバータの役割も果たします．

ispMACH4000シリーズはispM4Aシリーズの後継シリーズです．低電圧ロジックに対応し，デバイスの省電力化，高速化が図られています．アーキテクチャもispM4Aに近くチップの電源電圧によって，

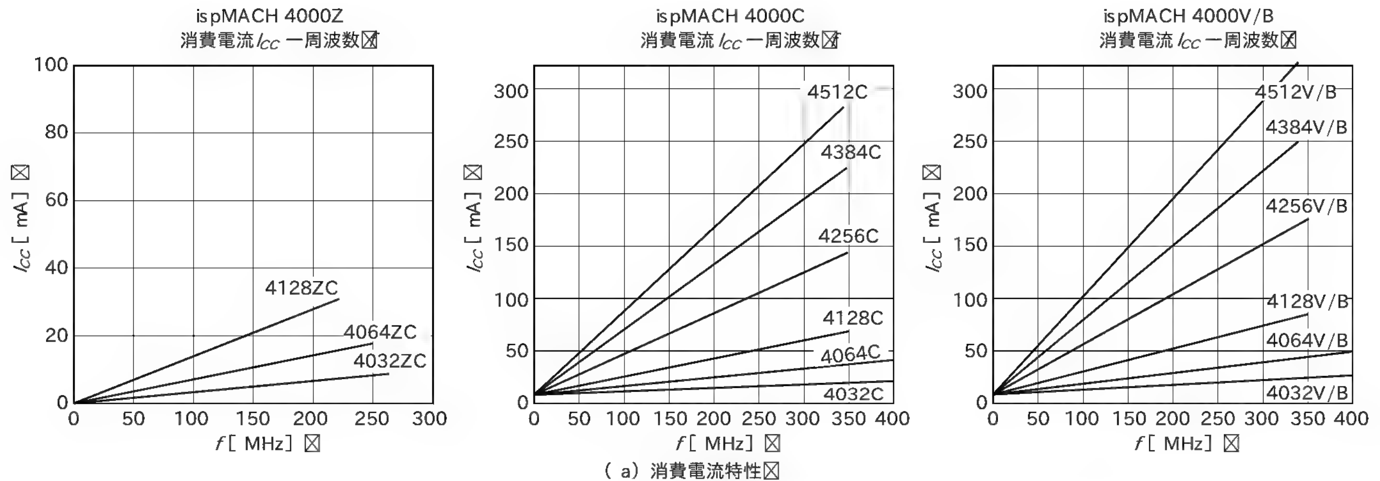
3.3V	ispMACH4000V
2.5V	ispMACH4000B
1.8V	ispMACH4000C
1.8V	ispMACH4000Z

のサブファミリがラインナップされています．

最後のサフィクスであるV，B，C，Zはチップの電源電圧を表しています．ZサフィクスのispMACH4000Zはクロック静止

表1[1] ラティスセミコンダクター社のCPLD製品ラインナップ

ファミリ		電源電圧	集積度	速度		I/O数	ロジック・	メモリ・ビット	PLL
アーキテクチャ	品番		マクロセル数	t_{PD}(ns)	F_{max}(MHz)		ブロック入力数	(K)	
ispXPLD5000	ispXPLD5000MC	1.8 V	256～1024	3.5	300	141～381	68	64～512	2
	ispXPLD5000MB	2.5 V	256～1024	3.5	300	141～381	68	64～512	2
	ispXPLD5000MV	3.3 V	256～1024	3.5	300	141～381	68	64～512	2
ispMACH5000	ispMACH5000B	2.5 V	128～512	3.5	275	92～256	68	－	－
	ispMACH5000VG	3.3 V	768～1024	5	178	196～384	68	－	2
	ispMACH5000VE	3.3 V	128～512	5	180	72～256	68	－	－
ispMACH4000	ispMACH4000Z	1.8 V	32～256	3.5	267	32～128	36	－	－
	ispMACH4000C	1.8 V	32～512	2.5	400	30～208	36	－	－
	ispMACH4000B	2.5 V	32～512	2.5	400	30～208	36	－	－
	ispMACH4000V	3.3 V	32～512	2.5	400	30～208	36	－	－
ispMACH4A	ispMACH4A3	3.3 V	32～512	5	182	32～256	33～36	－	－
	ispMACH4A5	5.0 V	32～256	5	182	32～128	33～36	－	－



(a) 消費電流特性

時の消費電力が約50μ～190μAのゼロ・パワー・デバイスです．5V電源ファミリはありませんが，ispMACH4000Vの入出力は「5Vトレラント」です．5Vロジック回路の信号を入力として受けることができるので，5V混在システムにも使うことができます．

図11はispMACH4000シリーズの〔消費電流－周波数〕特性です．いずれのサブファミリも動作周波数に比例して消費電力が増加します．サフィクスV，B，Cのデバイスはクロック周波数＝0MHzのときも電源電流が流れますが，4000Zは200μA以下と，ほとんどゼロになります．これが「ゼロ・パワー」と呼ばれるゆえんです．

携帯電話やデジカメなどの急速な普及により，CPU，メモリ，周辺インターフェース・チップも急速に電源電圧の低電圧化が進んでいます．4000シリーズはこの流れに対応したCPLDファミリです．

このファミリはチップの電源電圧とは別に入出力電圧，ロジック仕様を選択できます．3.3V，2.5V，1.8Vとデバイスが混在したシステムではレベル・コンバータの役割も果たすことができます（図12）．

型名	A	B
ispMACH 4032V/B	11.3	0.010
ispMACH 4032C	1.3	0.010
ispMACH 4064V/B	11.5	0.010
ispMACH 4064C	1.5	0.010
ispMACH 4128V/B	11.5	0.011
ispMACH 4128C	1.5	0.011
ispMACH 4256V/B	12	0.011
ispMACH 4256C	2	0.011
ispMACH 4384V/B	12.5	0.013
ispMACH 4384C	2.5	0.013
ispMACH 4512V/B	13	0.013
ispMACH 4512C	3	0.013
ispMACH 4032ZC	0.010	0.010
ispMACH 4064ZC	0.011	0.010
ispMACH 4128ZC	0.012	0.010
ispMACH 4256ZC	–	–

消費電流 $I_{CC} = A + B \cdot N \cdot f_{max} \cdot AF + I_{CCO}$

I_{CC}：ispMACH4000シリーズの算定係数
A：定常電流係数（mA）
B：マクロセル電力係数
N：使用マクロセル数
f_{max}：最大動作周波数（MHz）
AF：ノード利用係数
I_{CCO}：出力ピン消費電流

(b) 消費電流算定係数

図11[1] ispMACH4000シリーズの消費電流と動作周波数の関係

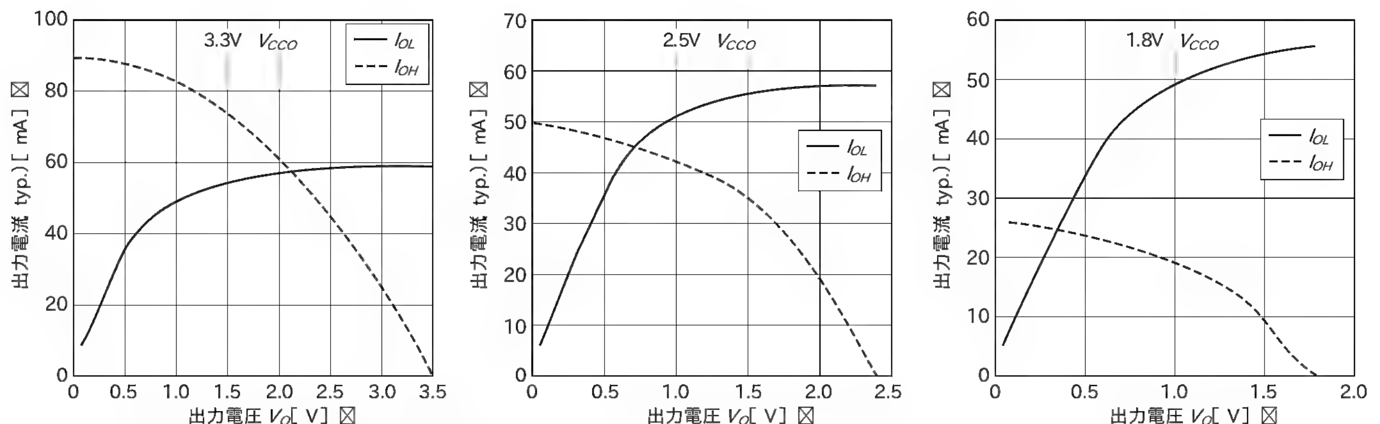


図12[1] ispMACH4000シリーズのI/O出力電流-電圧特性

ispMACH5000シリーズはロジック・ブロック入力数が68と4000シリーズより大規模なロジック構成に適したファミリです．

ispMACHの各シリーズはEEPROM内蔵のCPLDでパソコン上で開発したロジック・データをISP（イン・システム・プログラミング）によりケーブル1本でチップに書き込むことができます．書き込まれたデータは，チップ内のEEPROMに記憶されるので電源を切っても消えません．

FPGAのように，外部にコンフィグレーション用のEEPROMを用意して，電源投入時ごとにロジック・データをロードする必要はありません．

ispXPLD5000シリーズは，ロジック・セルにSRAMを採用したデバイスです．SRAM上のデータは電源を切ると消えるので，電源投入時にロジック・データをロードする必要があります．しかし，XPLDはチップ内部にコンフィグレーション用のEEPROMを内蔵していて電源投入時に自動的にロードされます．ユーザから見ると従来のispMACHと同じ使い勝手で，高機能・高集積度のデバイスが手に入ります．

● ispMACH4000シリーズの製品ラインナップ

表2はispMACH4000シリーズの製品ラインナップです．チップの電源電圧によるラインナップ（V，B，C，Z）のほか，内蔵マクロセルの数，パッケージにより多くの製品が用意されています．

ROBO-02に使ったLC4256V-10T100Iは，

電源電圧　3.3V
遅延時間（t_{PD}）　10ns
マクロセル　256個
パッケージ　100ピンTQFP

の製品です（パッケージおよび遅延時間を含めた製品品番はispMACHをLCで置き換える）．

ispMACH4000シリーズのデバイスはチップの電源電圧とは別に，表3に示す入出力電圧特性を選択できます．この選択はデバイスのプログラミングとI/O電圧 V_{CCO} によって設定しま

表2[1] CPLD ispMACH4000ファミリの製品ラインナップ

	ispMACH 4032V/B/C	ispMACH 4064V/B/C	ispMACH 4128V/B/C	ispMACH 4256V/B/C	ispMACH 4384V/B/C	ispMACH 4512V/B/C
マクロセル	32	64	128	256	384	512
入出力ピン＋入力専用ピン	30+2/32+4	30+2/32+4/64+10	64+10/92+4/96+4	64+10/96+14/128+4/160+4	128+4/192+4	128+4/208+4
t_{PD}（ns）	2.5	2.5	2.7	3.0	3.5	3.5
t_S（ns）	1.8	1.8	1.8	2.0	2.0	2.0
t_{CO}（ns）	2.2	2.2	2.7	2.7	2.7	2.7
f_{max}（MHz）	400	400	333	322	322	322
電源電圧（V）	3.3/2.5/1.8	3.3/2.5/1.8	3.3/2.5/1.8	3.3/2.5/1.8	3.3/2.5/1.8	3.3/2.5/1.8
ピン/パッケージ仕様	44TQFP 48TQFP	44TQFP 48TQFP 100TQFP	100TQFP 128TQFP 144TQFP[1]	100TQFP 144TQFP[1] 176TQFP 256 fpBGA[2]	176TQFP 256 fpBGA	176TQFP 256 fpBGA

1）3.3V（4000V）
2）128ピンI/O and 160ピンI/O

（a）ispMACH4000V/B/C

	ispMACH 4032ZC	ispMACH 4064ZC	ispMACH 4128ZC	ispMACH 4256ZC
マクロセル	32	64	128	256
入出力ピン＋入力専用ピン	32+4	32+4/32+12/64+10	64+10/96+4	64+10/96+6/128+4
t_{PD}（ns）	3.5	3.7	4.2	5.0
t_S（ns）	2.2	2.5	2.7	3.0
t_{CO}（ns）	3.0	3.2	3.5	3.9
f_{max}（MHz）	267	250	220	200
電源電圧（V）	1.8	1.8	1.8	1.8
最大スタンバイ電流 I_{CC}（μA）	20	25	35	55
ピン/パッケージ仕様	48TQFP 56 csBGA	48TQFP 56 csBGA 100TQFP 132 csBGA	100TQFP 132 csBGA	100TQFP 132 csBGA 176TQFP

（b）ispMACH4000Zファミリ

表3[1] ispMACH4000シリーズの入出力直流電気的特性(ただし，下記推奨動作条件による)

標準ロジック	入力特性				出力特性			
	V_{IL}		V_{IH}		V_{OL}	V_{OH}	I_{OL}	I_{OH}
	min (V)	max (V)	min (V)	max (V)	max (V)	min (V)	(mA)	(mA)
LVTTL	−0.3	0.80	2.0	5.5	0.40	V_{CCO}−0.40	8.0	−4.0
					0.20	V_{CCO}−0.20	0.1	−0.1
LVCMOS 3.3	−0.3	0.80	2.0	5.5	0.40	V_{CCO}−0.40	8.0	−4.0
					0.20	V_{CCO}−0.20	0.1	−0.1
LVCMOS 2.5	−0.3	0.70	1.70	3.6	0.40	V_{CCO}−0.40	8.0	−4.0
					0.20	V_{CCO}−0.20	0.1	−0.1
LVCMOS 1.8 (4000V/B)	−0.3	0.63	1.17	3.6	0.40	V_{CCO}−0.45	2.0	−2.0
					0.20	V_{CCO}−0.20	0.1	−0.1
LVCMOS 1.8 (4000C/Z)	−0.3	0.35V_{CC}	0.65V_{CC}	3.6	0.40	V_{CCO}−0.45	2.0	−2.0
					0.20	V_{CCO}−0.20	0.1	−0.1
PCI 3.3 (4000V/B)	−0.3	1.08	1.5	5.5	0.1V_{CCO}	0.9V_{CCO}	1.5	−0.5
PCI 3.3 (4000C/Z)	−0.3	0.3/3.3 (V_{CC}/1.8)	0.5/3.3 (V_{CC}/1.8)	5.5	0.1V_{CCO}	0.9V_{CCO}	1.5	−0.5

注) I/O推奨動作条件(I/O基準設定vs.推奨I/O電源電圧)

標準ロジック	I/O電圧V_{CCO}(V)	
	min	max
LVTTL	3.0	3.6
LVCMOS 3.3	3.0	3.6
Extended LVCMOS 3.3	2.7	3.6
LVCMOS 2.5	2.3	2.7
LVCMOS 1.8	1.65	1.95
PCI 3.3	3.0	3.6

す．表3のI/O推奨動作条件は各入出力電圧特性を選択したときのV_{CCO}電圧を示しています．

たとえば，デバイスの電源電圧3.3Vで入出力電圧特性LVCMOS2.5を選択する場合は，

電源電圧(V_{CC})　3.3V
I/O電圧(V_{CCO})　2.5V

とします．

表4はROBO-02に使ったLC4256V-10T100Iのピン配列表です．I/O電圧(V_{CCO})は二つのバンク(Bank 0, Bank 1)ごとに

表4[1] ispMACH4256V/B/C(100ピンTQFPパッケージ)のロジック配線表(*印のピンは入力のみ)

ピン番号	Bank番号	ispMACH4256V/B/C GLB/MC/Pad	ピン番号	Bank番号	ispMACH4256V/B/C GLB/MC/Pad	ピン番号	Bank番号	ispMACH4256V/B/C GLB/MC/Pad	ピン番号	Bank番号	ispMACH4256V/B/C GLB/MC/Pad
1	−	GND	26	−	GND	51	−	GND	76	−	GND
2	−	TDI	27*	0	I	52	−	TMS	77*	1	I
3	0	C12	28	0	G12	53	1	K12	78	1	O12
4	0	C10	29	0	G10	54	1	K10	79	1	O10
5	0	C6	30	0	G6	55	1	K6	80	1	O6
6	0	C2	31	0	G2	56	1	K2	81	1	O2
7	0	GND (Bank 0)	32	0	GND (Bank 0)	57	1	GND (Bank 1)	82	1	GND (Bank 1)
8	0	D12	33	0	V_{CCO} (Bank 0)	58	1	L12	83	1	V_{CCO} (Bank 1)
9	0	D10	34	0	H12	59	1	L10	84	1	P12
10	0	D6	35	0	H10	60	1	L6	85	1	P10
11	0	D4	36	0	H6	61	1	L4	86	1	P6
12*	0	I	37	0	H2	62*	1	I	87	1	P2/OE1
13	0	V_{CCO} (Bank 0)	38	0	CLK1/I	63	1	V_{CCO} (Bank 1)	88	1	CLK3/I
14	0	E4	39	1	CLK2/I	64	1	M4	89	0	CLK0/I
15	0	E6	40	−	V_{CC}	65	1	M6	90	−	V_{CC}
16	0	E10	41	1	I2	66	1	M10	91	0	A2/GOE0
17	0	E12	42	1	I6	67	1	M12	92	0	A6
18	0	GND (Bank 0)	43	1	I10	68	1	GND (Bank 1)	93	0	A10
19	0	F2	44	1	I12	69	1	N2	94	0	A12
20	0	F6	45	1	V_{CCO} (Bank 1)	70	1	N6	95	0	V_{CCO} (Bank 0)
21	0	F10	46	1	GND (Bank 1)	71	1	N10	96	0	GND (Bank 0)
22	0	F12	47	1	J2	72	1	N12	97	0	B2
23*	0	I	48	1	J6	73*	1	I	98	0	B6
24	−	TCK	49	1	J10	74	−	TDO	99	0	B10
25	−	V_{CC}	50	1	J12	75	−	V_{CC}	100	0	B12

別の電圧を選択することも可能です．

● ispMACH4000シリーズの開発ツール

ラティスセミコンダクター社のCPLD開発ツールとして，

- ispLEVER Advance $ 1,295
- ispLEVER Base $ 495
- ispLEVER Starter（6か月試用版） 無償

が用意されています．いずれもオンライン価格です．AdvanceとBaseにはダウンロード・ケーブル（図13）が付いています．

ispLEVER Starter（6か月試用版）は，ラティスセミコンダクター社のWebからダウンロードしてライセンス登録をすればすぐ使えます．評価版にはダウンロード・ケーブルが付いていませんが，オンライン・ストアで$ 65で購入できます．

ラティスセミコンダクター社は数年前にVantis社を買収しましたが，図13のダウンロード・ケーブルはその歴史を踏襲しています．図（a）の10ピン・タイプはVantis社，図（b）の8ピン・タイプは旧ラティスセミコンダクター社のダウンロード・ケーブルです．買収によりどちらのユーザにも不都合が起こらないように，書き込みツールは両ケーブルをサポートしています．筆者はVantis社時代から自作のダウンロード・ケーブルを使っていますが問題なく書き込めます．

ispLEVERの各製品は図14の論理設計フローで開発します．

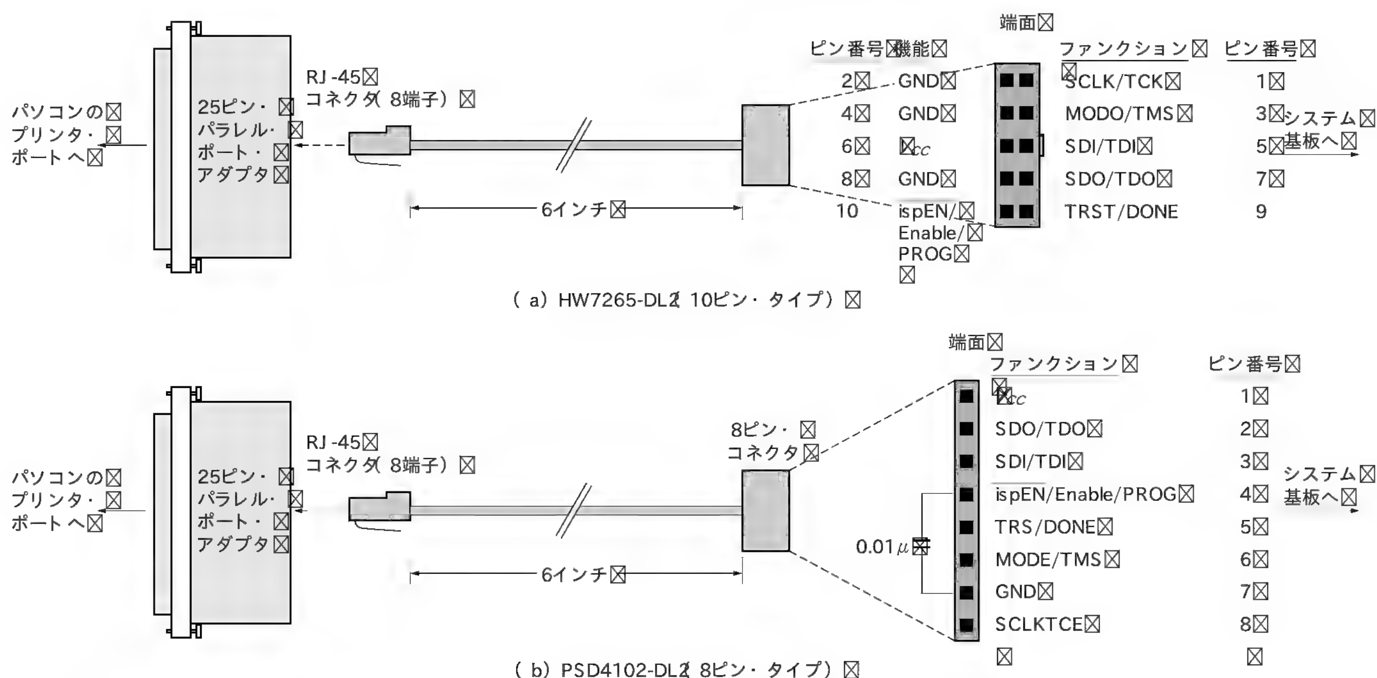


図13 ispダウンロード・ケーブル

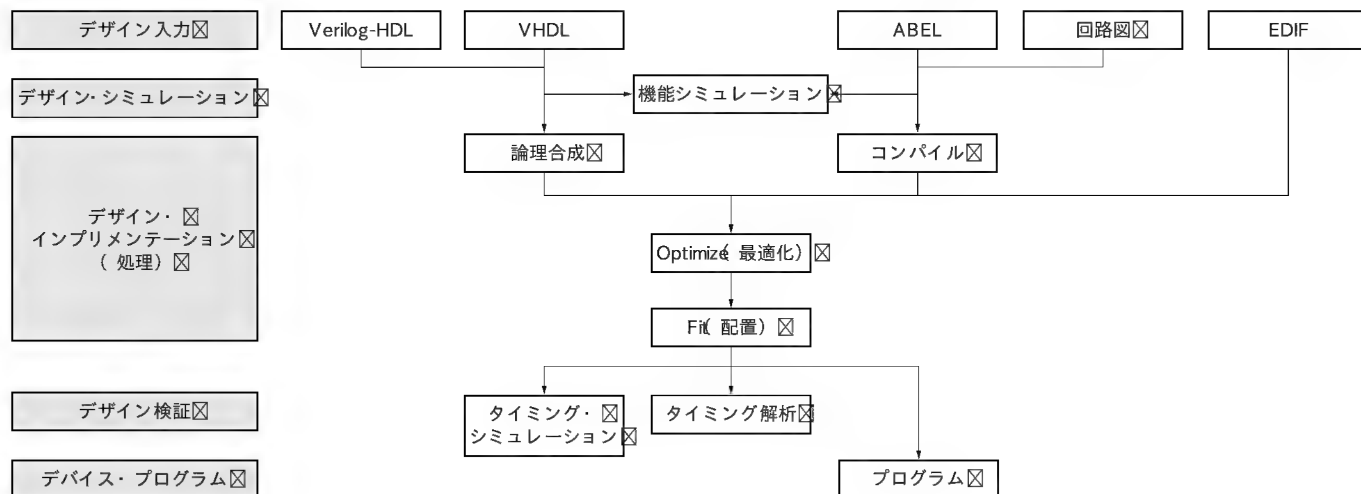


図14 CPLDの論理設計のフロー

開発ツールispLEVERによるCPLD開発手順

リスト1はROBO-02のPWM信号発生回路の論理設計データです．2個のispMACH4256のうちの1個で記述言語はABELです．

この論理回路をCPLDのヒューズ・データ(JEDECフォーマット)に変換してCPLDに書き込むまでの手順を簡単に紹介します．

まず，

- 開発ツール　ispLEVER Starter(試用版)
- 書き込みツール　ispVMSystem(無償)

をラティスセミコンダクター社のWebサイトからダウンロードしてWindowsパソコンにインストールします．

図15に示すようにispLEVERを起動し，

File → New Project

で新規プロジェクトを作成します．図16(p.93)の画面が表示されるので，プロジェクト名を指定し，Project Typeにより設計言語を指定します．

リスト1　二足歩行ロボット制御ボードROBO-02のPWM回路ロジック(PWM4256A.abl)

```
MODULE counter

TITLE '10 bit preloadable up counter'

"Constants
    c,x = .c.,.x.;

"Inputs
    clk0        pin     89;
    !reset      pin     88;
    A0..A7      pin     8,9,10,11,12,14,15,16;
    A16..A17    pin     19,20;
    D0..D7      pin     28,29,30,31,34,35,36,37;
    !CS0        pin     62;
    !WRL        pin     73;

"Node
    SL0..SL23   node    istype 'com';
"    SL256      node    istype 'com';
    PWON        node    istype 'reg,buffer';
    WRL0        node    istype 'reg,buffer';
    WRLC        node    istype 'reg,buffer';
    q18..q0     node    istype 'reg,buffer';
    PW09..PW00  node    istype 'reg,buffer';
    PW19..PW10  node    istype 'reg,buffer';
    PW29..PW20  node    istype 'reg,buffer';
    PW39..PW30  node    istype 'reg,buffer';
    PW49..PW40  node    istype 'reg,buffer';
    PW59..PW50  node    istype 'reg,buffer';
    PW69..PW60  node    istype 'reg,buffer';
    PW79..PW70  node    istype 'reg,buffer';
    PW89..PW80  node    istype 'reg,buffer';
    PW99..PW90  node    istype 'reg,buffer';
    PW109..PW100 node   istype 'reg,buffer';
    PW119..PW110 node   istype 'reg,buffer';

"Outputs
    PWM0..PWM11 pin     42,43,48,49,54,55,65,66,70,71,79,80
                                       istype 'reg_sr,buffer';
    !CS1        pin     5       istype  'com';
    !CS2        pin     6       istype  'com';
"    FA16..FA18 pin             istype  'reg,buffer';

"Sets
    address = [A7..A0];
    data   = [D7..D0];
    dataH   = [D1..D0];
"    dataF   = [D2..D0];

    count = [q18..q0];
    PW     = [q15..q5];

"    FA     = [FA18..FA16];

    PW0L   = [PW07..PW00];
    PW0H   = [PW09..PW08];
    PW0    = [1,PW09,PW08,PW07,PW06,PW05,PW04,PW03,
                                          PW02,PW01,PW00];

    PW1L   = [PW17..PW10];
    PW1H   = [PW19..PW18];
    PW1    = [1,PW19,PW18,PW17,PW16,PW15,PW14,PW13,
                                          PW12,PW11,PW10];

    PW2L   = [PW27..PW20];
    PW2H   = [PW29..PW28];
    PW2    = [1,PW29,PW28,PW27,PW26,PW25,PW24,PW23,
                                          PW22,PW21,PW20];

    PW3L   = [PW37..PW30];
    PW3H   = [PW39..PW38];
    PW3    = [1,PW39,PW38,PW37,PW36,PW35,PW34,PW33,
                                          PW32,PW31,PW30];

    PW4L   = [PW47..PW40];
    PW4H   = [PW49..PW48];
    PW4    = [1,PW49,PW48,PW47,PW46,PW45,PW44,PW43,
                                          PW42,PW41,PW40];

    PW5L   = [PW57..PW50];
    PW5H   = [PW59..PW58];
    PW5    = [1,PW59,PW58,PW57,PW56,PW55,PW54,PW53,
                                          PW52,PW51,PW50];

    PW6L   = [PW67..PW60];
    PW6H   = [PW69..PW68];
    PW6    = [1,PW69,PW68,PW67,PW66,PW65,PW64,PW63,
                                          PW62,PW61,PW60];

    PW7L   = [PW77..PW70];
    PW7H   = [PW79..PW78];
    PW7    = [1,PW79,PW78,PW77,PW76,PW75,PW74,PW73,
                                          PW72,PW71,PW70];

    PW8L   = [PW87..PW80];
    PW8H   = [PW89..PW88];
    PW8    = [1,PW89,PW88,PW87,PW86,PW85,PW84,PW83,
                                          PW82,PW81,PW80];

    PW9L   = [PW97..PW90];
    PW9H   = [PW99..PW98];
    PW9    = [1,PW99,PW98,PW97,PW96,PW95,PW94,PW93,
                                          PW92,PW91,PW90];

    PW10L   = [PW107..PW100];
    PW10H   = [PW109..PW108];
    PW10R    = [1,PW109,PW108,PW107,PW106,PW105,PW104,
                                 PW103,PW102,PW101,PW100];

    PW11L   = [PW117..PW110];
    PW11H   = [PW119..PW118];
    PW11R    = [1,PW119,PW118,PW117,PW116,PW115,PW114,
                                 PW113,PW112,PW111,PW110];

Equations

    SL0=(address == ^h00) & WRLC;
    SL1=(address == ^h01) & WRLC;
    SL2=(address == ^h02) & WRLC;
    SL3=(address == ^h03) & WRLC;
    SL4=(address == ^h04) & WRLC;
    SL5=(address == ^h05) & WRLC;
    SL6=(address == ^h06) & WRLC;
    SL7=(address == ^h07) & WRLC;
```

リスト 1　二足歩行ロボット制御ボード ROBO-02のPWM回路ロジック(PWM4256A.abl)(つづき)

```
    SL8=(address == ^h08) & WRLC;
    SL9=(address == ^h09) & WRLC;
    SL10=(address == ^h0A) & WRLC;
    SL11=(address == ^h0B) & WRLC;

    SL12=(address == ^h0C) & WRLC;
    SL13=(address == ^h0D) & WRLC;
    SL14=(address == ^h0E) & WRLC;
    SL15=(address == ^h0F) & WRLC;
    SL16=(address == ^h10) & WRLC;
    SL17=(address == ^h11) & WRLC;
    SL18=(address == ^h12) & WRLC;
    SL19=(address == ^h13) & WRLC;
    SL20=(address == ^h14) & WRLC;
    SL21=(address == ^h15) & WRLC;
    SL22=(address == ^h16) & WRLC;
    SL23=(address == ^h17) & WRLC;

"    SL256=(address == ^hFF) & WRLC;

    CS1 = !A17 & CS0;
    CS2 = A17 & !A16 & CS0;

"FA
"    FA := dataF & SL256
"         # FA & !SL256;
"    FA.clk = clk0;

    PWON := (count == 12800);
    PWON.clk=clk0;

    WRL0:=WRL;
    WRL0.clk=clk0;

    WRLC:=A17 & A16 & CS0 & WRL & !WRL0;
    WRLC.clk=clk0;

    count := (count.fb + 1);
    count.clk = clk0;
    count.ar = reset;

"PWM0
    PW0L := data & SL1
         # PW0L & !SL1;
    PW0L.clk = clk0;

    PW0H := dataH & SL0
         # PW0H & !SL0;
    PW0H.clk = clk0;

    PWM0.r = (PW == PW0);
    PWM0.s = PWON;
    PWM0.clk = clk0;

"PWM1
    PW1L := data & SL3
         # PW1L & !SL3;
    PW1L.clk = clk0;

    PW1H := dataH & SL2
         # PW1H & !SL2;
    PW1H.clk = clk0;

    PWM1.r = (PW == PW1);
    PWM1.s = PWON;
    PWM1.clk = clk0;

"PWM2
    PW2L := data & SL5
         # PW2L & !SL5;
    PW2L.clk = clk0;

    PW2H := dataH & SL4
         # PW2H & !SL4;
    PW2H.clk = clk0;

    PWM2.r = (PW == PW2);
    PWM2.s = PWON;
    PWM2.clk = clk0;

"PWM3
    PW3L := data & SL7
         # PW3L & !SL7;
    PW3L.clk = clk0;

    PW3H := dataH & SL6
         # PW3H & !SL6;
    PW3H.clk = clk0;

    PWM3.r = (PW == PW3);
    PWM3.s = PWON;
    PWM3.clk = clk0;

"PWM4
    PW4L := data & SL9
         # PW4L & !SL9;
    PW4L.clk = clk0;

    PW4H := dataH & SL8
         # PW4H & !SL8;
    PW4H.clk = clk0;

    PWM4.r = (PW == PW4);
    PWM4.s = PWON;
    PWM4.clk = clk0;

"PWM5
    PW5L := data & SL11
         # PW5L & !SL11;
    PW5L.clk = clk0;

    PW5H := dataH & SL10
         # PW5H & !SL10;
    PW5H.clk = clk0;

    PWM5.r = (PW == PW5);
    PWM5.s = PWON;
    PWM5.clk = clk0;

"PWM6
    PW6L := data & SL13
         # PW6L & !SL13;
    PW6L.clk = clk0;

    PW6H := dataH & SL12
         # PW6H & !SL12;
    PW6H.clk = clk0;

    PWM6.r = (PW == PW6);
    PWM6.s = PWON;
    PWM6.clk = clk0;

"PWM7
    PW7L := data & SL15
         # PW7L & !SL15;
    PW7L.clk = clk0;

    PW7H := dataH & SL14
         # PW7H & !SL14;
    PW7H.clk = clk0;

    PWM7.r = (PW == PW7);
    PWM7.s = PWON;
    PWM7.clk = clk0;

"PWM8
    PW8L := data & SL17
         # PW8L & !SL17;
    PW8L.clk = clk0;

    PW8H := dataH & SL16
         # PW8H & !SL16;
    PW8H.clk = clk0;

    PWM8.r = (PW == PW8);
    PWM8.s = PWON;
    PWM8.clk = clk0;

"PWM9
    PW9L := data & SL19
         # PW9L & !SL19;
    PW9L.clk = clk0;

    PW9H := dataH & SL18
         # PW9H & !SL18;
    PW9H.clk = clk0;

    PWM9.r = (PW == PW9);
    PWM9.s = PWON;
    PWM9.clk = clk0;

"PWM10
    PW10L := data & SL21
         # PW10L & !SL21;
    PW10L.clk = clk0;

    PW10H := dataH & SL20
         # PW10H & !SL20;
    PW10H.clk = clk0;

    PWM10.r = (PW == PW10R);
    PWM10.s = PWON;
    PWM10.clk = clk0;

"PWM11
    PW11L := data & SL23
         # PW11L & !SL23;
    PW11L.clk = clk0;

    PW11H := dataH & SL22
         # PW11H & !SL22;
    PW11H.clk = clk0;

    PWM11.r = (PW == PW11R);
    PWM11.s = PWON;
    PWM11.clk = clk0;

END
```



図 15　開発ツール ispLEVER を起動する

ispLEVERは設計言語としてABEL，VHDL，Verilog-HDLを使うことができます．筆者の場合はABELですから図のように

Schematic/ABEL

を選択します．

するとデフォルトのデバイス名が選択されるので，これをダブル・クリックします．すると図17のデバイス選択画面が表示されるので，

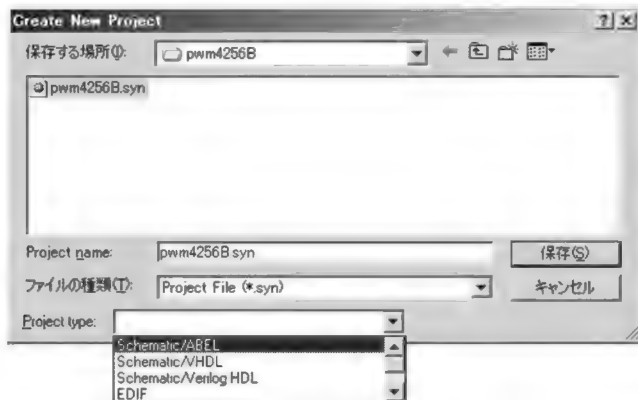


図16　New Projectを作成する…プロジェクト名とプロジェクト・タイプを指定

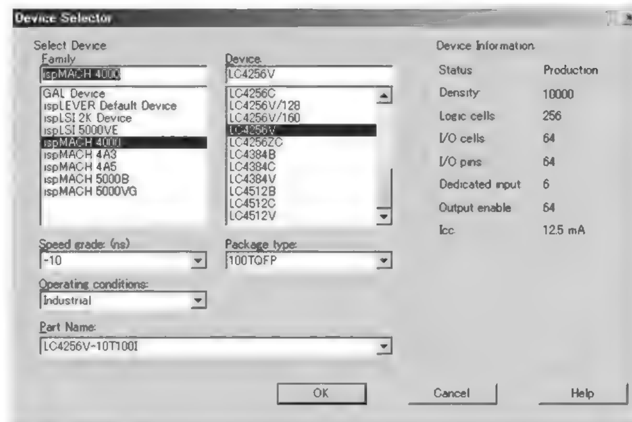


図17　デバイス・セレクト画面でデバイスを選択する

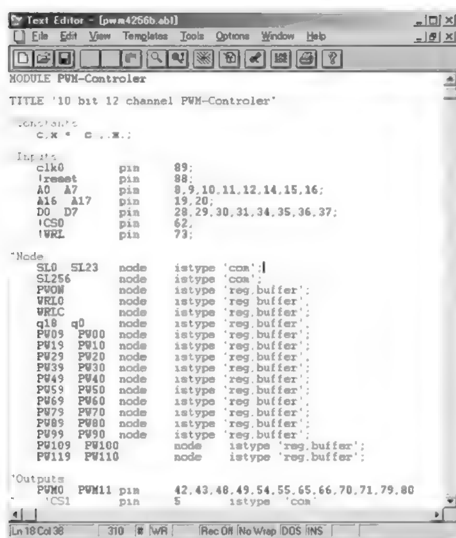


図18　テキスト・エディタを起動してソース・ファイルを作成

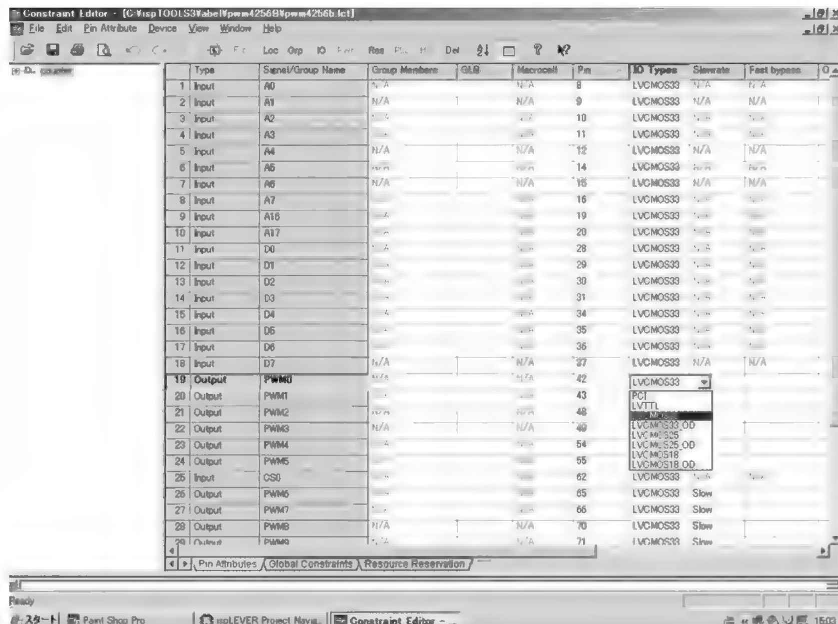


図19　コンストレイント・エディタ画面でピン番号，I/Oタイプ，スルーレートなどを指示する

```
Device Family     =ispMACH 4000
Device            =LC4256V
Package type      =100TQFP
Speed grade       =10ns
```

を選択します．

次にデバイス名の上にマウスを置いて右ボタンをクリックするとプルダウン・メニューが出るので「New:新規」もしくは「Import:追加」で設計データ・ファイル（PWM4256B.ABL）を登録します．設計データ・ファイル名をダブル・クリックするとテキスト・エディタが開きます．論理記述の変更はこの画面で行います（図18）．

Source in Projectウィンドウでデバイス名をマウスで選択すると右側の［Processes for current source］ウインドウに，

［Constraint Editor］

が表示されます．

これをダブルクリックすると図19の「コンストレイント・エディタ画面」が開きます．この画面は論理回路のコンパイル&フィッティング条件を指定するためのものです．

図19に示すようにピンごとにI/Oタイプやスルーレートを指定することができます．また［Pin Attribute］を使って，

［Pin Attribute］→［Location Assignment］

により，

- ピン番号と信号の割り付け
- マクロセルの割り付け

を行うこともできます（図20）．

デバイス・リソースへの割り付けはフィッタの自動割り付けに任せることも可能ですし，必要に応じてユーザが指定することも可能です．

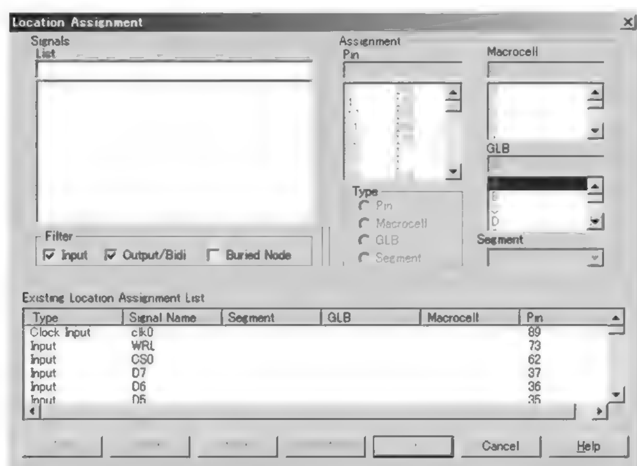


図20　コンストレイント・エディタで Location Assignment によりピン番号，マクロセルを割り付ける

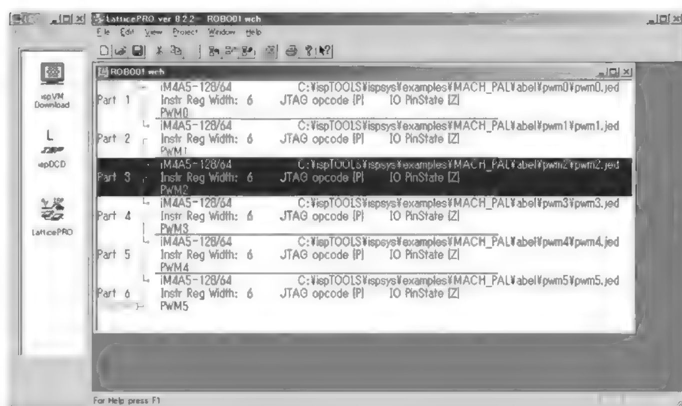


図23　LatticePRO ver.8.2 画面…カスケード接続されたデバイスを順に登録する（これは第2章で紹介した ROBO-01A の例）

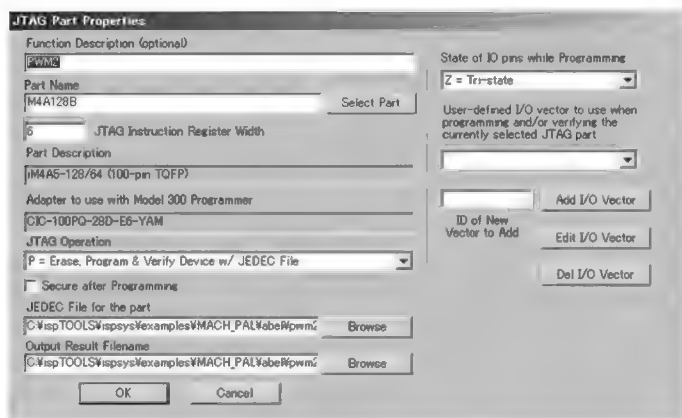


図24　JTAGパーツ・プロパティ画面でパーツを登録する

必要に応じてリソースの割り付けを行った後，〔Fit Design〕をダブル・クリックするとフィッティングが始まります．その結果，

Completed successfully

と表示されれば設計は完了です（図21）．設計データに論理的なミスがなくても，実際のデバイスに入りきらない場合はエラーが表示されます．



図21　フィッティング完了



図22　ISPプログラミング・ツール ispVM システムを起動し，LatticePRO を選択する

設計データをデバイス（CPLD）に書き込む手順

設計が完了したら JEDEC データをデバイスに書き込みます．ラティスセミコンダクター社の CPLD はすべて ISP（イン・システム・プログラミング）が可能です．基板に実装した状態で ISP ケーブルを使ってパソコンからデータを送って書き込みます．

まず，図13の ISP ケーブルによりパソコンのプリンタ・ポートとデバイス実装基板とを接続します．基板の電源を入れて書き込みツール ispVM System を起動すると図22の画面が開くので，〔LatticePRO〕をダブルクリックします．

すると図23の画面が開くので，カスケード接続されたデバイスを順に登録します．図23は ROBO-01A の6個の M4A 5128/64 を登録した状態です．

各デバイスは図24に示す JTAG パーツ・プロパティ画面で，

- パーツ名
- JEDEC ファイル名
- JTAG オペレーション

などを指定した上で〔GO〕を押すとデバイス1から順に書き込み（ISP プログラミング）が始まります．

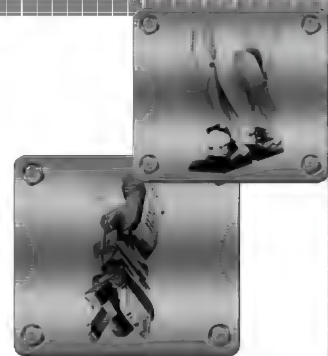
引用＊文献

（1）＊ラティスセミコンダクター，ispMACH4000V/B/C/Z Family データシート，2003年11月

よしだ・こうさく

Appendix 2

CPLD の開発言語はなにを使うべきか… ABEL vs VHDL



吉田 幸作/井倉 将実

筆者のフィールド・プログラマブル・デバイス歴は，バイポーラのPAL16R6からですから20年以上になります．デバイスの急速な進歩に合わせて開発環境も大きく変わってきました．PALASMからABEL，そしてVerilog-HDLやVHDLへと開発言語を変えることは技術者にとって大きなハードルがあります．

今回のPWM信号発生回路の開発も使い慣れたABELを使いました．「CPLD程度の集積度の開発にはABELのほうがむだがない」というのが筆者の口癖ですが，CycloneクラスのFPGAの開発にはVHDLに頼らざるをえないことは明白です．

VHDLのプロ（井倉将実氏）に「ABELで書いたソースをVHDLに書き直して，Altera社のCycloneとXilinx社のSpartan-IIに入れ込む」という依頼をしたら結果が1時間40分後に届きました．

リストAはその記述リストです．試しにラティスセミコンダクター社のispLEVERを使ってispMACH4256に組み込んでみました．フィッティングも無事終了しました．

まだ動作テストは行っていませんが，「ABELのほうが実装効率が…」という筆者の主張はあやしくなってきました．開発ツールはあくまでも道具なので，使いやすいものを使えばよいと思うのですが…．

よしだ・こうさく/いくら・まさみ　来栖川電工（有）

リストA　リスト1のPWM4256A.ablをVHDL記述に変更（pwm4256.vhd）

```
--
--  FILE NAME   : PWM4256A
--                    12bit-data width PWM controller
--
--  Original           : Yoshida-san's design
--
--       HDL-Design    : Masami IKURA
--                     / KURUSUGAWA Electronics industry Inc,.
--                     URL : http://www.kurusugawa-ele.co.jp
--
--  Release            : 2004, Feb, 11th
--
-- < NOTE >
--       TOP-Module    : PWM4256A
--                |
--                ------ PWM_UNIT.vhd
--

library ieee ;
use ieee.std_logic_unsigned.all ;
use ieee.std_logic_arith.all ;
use ieee.std_logic_1164.all ;

entity PWM4256A is
        generic (
                PWM_WIDTH    : integer := 12   -- PWM unit = 12.
        );
        port (
                clk0      : in     std_logic ;
                nreset    : in     std_logic ;

                A         : in     std_logic_vector(7 downto 0) ;
                A17, A16  : in     std_logic ;
                D         : in     std_logic_vector(7 downto 0) ;
                nCS0      : in     std_logic ;
                nWRL      : in     std_logic ;

                PWM       : out  std_logic_vector(
                                          PWM_WIDTH-1 downto 0) ;
                nCS1, nCS2   : out  std_logic
        );
end PWM4256A ;

architecture RTL of PWM4256A is

-- ******** --
component PWM_UNIT is
        port (
                clk0     : in     std_logic ;
                reset    : in     std_logic ;

                DIN      : in     std_logic_vector(7 downto 0) ;
                HLD_H    : in     std_logic ;
                HLD_L    : in     std_logic ;
                PW       : in     std_logic_vector(15 downto 5) ;
                PWON     : in     std_logic ;

                PWM_O    : out    std_logic
        ) ;
end component;

        signal    reset       : std_logic ;

        signal    address     : std_logic_vector(7 downto 0) ;

        signal    data        : std_logic_vector(7 downto 0) ;
        signal    dataH       : std_logic_vector(1 downto 0) ;

        signal    count       : std_logic_vector(18 downto 0) ;
        signal    PW          : std_logic_vector(15 downto 5) ;
        signal    WRL0, WRLC  : std_logic ;
        signal    PWON        : std_logic ;

-- **** Declare ADDRESS-DECODE nodes **** --
        signal    SL          : std_logic_vector(23 downto 0) ;

begin

--
        reset      <= not nreset ;

-- Address decoder
        address    <= A(7 downto 0) ;

        SL(0)    <= '1'    when ( address = x"00" and
                                   WRLC = '1' ) else '0' ;
        SL(1)    <= '1'    when ( address = x"01" and
                                   WRLC = '1' ) else '0' ;
        SL(2)    <= '1'    when ( address = x"02" and
                                   WRLC = '1' ) else '0' ;
        SL(3)    <= '1'    when ( address = x"03" and
                                   WRLC = '1' ) else '0' ;
        SL(4)    <= '1'    when ( address = x"04" and
                                   WRLC = '1' ) else '0' ;
        SL(5)    <= '1'    when ( address = x"05" and
                                   WRLC = '1' ) else '0' ;
        SL(6)    <= '1'    when ( address = x"06" and
                                   WRLC = '1' ) else '0' ;
        SL(7)    <= '1'    when ( address = x"07" and
                                   WRLC = '1' ) else '0' ;
        SL(8)    <= '1'    when ( address = x"08" and
                                   WRLC = '1' ) else '0' ;
        SL(9)    <= '1'    when ( address = x"09" and
                                   WRLC = '1' ) else '0' ;
        SL(10)   <= '1'    when ( address = x"0a" and
                                   WRLC = '1' ) else '0' ;
```

リスト A　リスト 1 の PWM4256A.abl を VHDL 記述に変更（pwm4256.vhd）（つづき）

```
        SL(11)  <= '1'    when ( address = x"0b" and
                                WRLC = '1' ) else '0' ;

        SL(12)  <= '1'   when ( address = x"0c" and
                                WRLC = '1' ) else '0' ;
        SL(13)  <= '1'   when ( address = x"0d" and
                                WRLC = '1' ) else '0' ;
        SL(14)  <= '1'   when ( address = x"0e" and
                                WRLC = '1' ) else '0' ;
        SL(15)  <= '1'   when ( address = x"0f" and
                                WRLC = '1' ) else '0' ;
        SL(16)  <= '1'   when ( address = x"10" and
                                WRLC = '1' ) else '0' ;
        SL(17)  <= '1'   when ( address = x"11" and
                                WRLC = '1' ) else '0' ;
        SL(18)  <= '1'   when ( address = x"12" and
                                WRLC = '1' ) else '0' ;
        SL(19)  <= '1'   when ( address = x"13" and
                                WRLC = '1' ) else '0' ;
        SL(20)  <= '1'   when ( address = x"14" and
                                WRLC = '1' ) else '0' ;
        SL(21)  <= '1'   when ( address = x"15" and
                                WRLC = '1' ) else '0' ;
        SL(22)  <= '1'   when ( address = x"16" and
                                WRLC = '1' ) else '0' ;
        SL(23)  <= '1'   when ( address = x"17" and
                                WRLC = '1' ) else '0' ;

        nCS1    <= '0'   when ( nCS0 = '0' and
                              A17 = '0' ) else '1' ;
        nCS2    <= '0'   when ( nCS0 = '0' and A17 = '1' and
                              A16 = '0' ) else '1' ;

        data     <= D(7 downto 0)  ;              --
        dataH    <= D(1 downto 0) ;               --

--
--       FA
--

FA_Equ : process ( clk0, reset )
begin
        if ( reset = '1' ) then
                count    <= ( others => '0' ) ;
        elsif ( clk0'event and clk0 = '1' ) then
                 if ( count = 12800 ) then
                        PWON  <= '1' ;
                 else
                        PWON  <= '0' ;
                 end if ;

                WRL0   <= nWRL ;
                WRLC   <= A17 and A16 and not nCS0 and
                          not nWRL and WRL0 ;  -- Edge detect.

                 count   <= count + 1 ;

        end if ;
end process ;

        PW      <= count(15 downto 5) ;

u1 : for i in 0 to PWM_WIDTH-1 generate
        uPWM_UNIT : PWM_UNIT
                 port map (
                        clk0      => clk0 ,
                        reset      => reset ,

                        DIN      => data ,
                        HLD_H   => SL(i*2 + 1) ,
                        HLD_L   => SL(i*2) ,
                        PW      => PW(15 downto 5) ,    -- 11bit
                        PWON   => PWON ,

                        PWM_O  => PWM(i)
                ) ;
end generate ;


end RTL


--
```

```
--  FILE NAME   : PWM4256A
--                            12bit-data width PWM controller
--
--  Original       : Yoshida-san's design
--
--        HDL-Design    : Masami IKURA
--                       / KURUSUGAWA Electronics industry Inc,.
--                       URL : http://www.kurusugawa-ele.co.jp/
--
-- Release              : 2004, Feb, 11th
--
-- < NOTE >
--        TOP-Module : PWM4256A
--                  |
--                  ------ PWM_UNIT.vhd
--

library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_unsigned.all ;
use ieee.std_logic_arith.all ;

entity PWM_UNIT is
        port (
                clk0    : in     std_logic ;
                reset   : in     std_logic ;

                DIN     : in     std_logic_vector(7 downto 0) ;
                HLD_H   : in     std_logic ;
                HLD_L   : in     std_logic ;
                PW      : in     std_logic_vector(15 downto 5) ;
                PWON    : in     std_logic ;

                PWM_O   : out    std_logic
       ) ;
end ;

architecture RTL of PWM_UNIT is

        signal    data_L    : std_logic_vector( 7 downto 0) ;
        signal    data_H    : std_logic_vector( 1 downto 0) ;

-- **** Declare PWM pulse parameter register **** --
        signal    PW_L      : std_logic_vector( 7 downto 0) ;
        signal    PW_H      : std_logic_vector( 9 downto 8) ;
        signal    PWM_Data  : std_logic_vector(10 downto 0) ;

        signal    iPW_O     : std_logic ;      -- internal node.

begin

        data_L           <= DIN(7 downto 0) ;
        data_H           <= DIN(1 downto 0) ;

-- PW[11:0] declare
        PWM_Data       <= '1' & PW_H & PW_L ;

-- ******************************** --
PWM_Equ : process ( clk0, reset )
begin
        if ( reset = '1' ) then
                PW_L   <= ( others => '0' ) ;
                PW_H   <= ( others => '0' ) ;
                iPW_O  <= '0' ;
        elsif ( clk0'event and clk0 = '1' ) then
                if ( HLD_L = '1' )    then
                        PW_L    <= data_L ;
                end if ;
                if ( HLD_H = '1' ) then
                        PW_H    <= data_H ;
                end if ;

                if ( PWON = '1' ) then
                        iPW_O   <= '1' ;
                elsif ( PW = PWM_Data ) then
                        iPW_O   <= '0' ;
                end if ;
        end if ;
end process ;

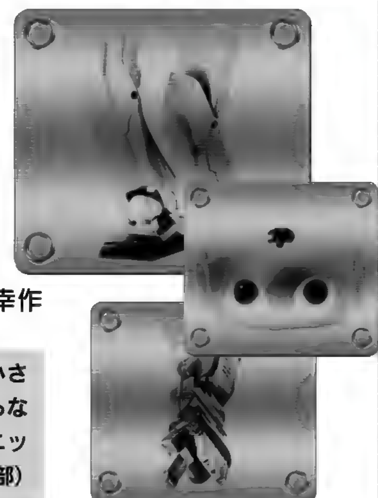
        PWM_O  <= iPW_O ;

end RTL ;
```

4. 市販部品と特注部品を使い分ける

ロボットの機構設計とサーボ・モータの選択

吉田 幸作



ロボットを作ると決めたときから，機構設計は欠かせない作業となる．電子制御部品をいかに小さく機構中に収めるか．CADで部品を作るときは「加工できない部品，組み立て不可能な部品」を作らないことが重要になってくる．最後にASIMOなどの高性能なロボットに多く使われているハーモニック・ドライブというモータの減速機についても紹介する．（編集部）

ロボットの機構部品の入手方法は…市販機構部品セット，特注部品，自作の道

ロボット開発の第一歩はロボットの機構設計とアクチュエータの入手です．方法は二つ，自分で作るか，購入するかです．購入する場合も既製品を購入する場合と注文して作ってもらう場合があります．

いちばん簡単な方法は既製品を購入する方法です．プロローグで紹介したロボットの中には「研究用プラットホーム」として販売されているものがあります．表1は入手可能な商品のリストです．

表1　市販されている二足歩行ロボット本体および機構部品セット

仕様 ＼ ロボット名		ロボット研究用プラットホーム(300万円以上)		
		KOZOH III	HOAP-2	nuvo(ぬーボー)
開発主体		ロボス(株)	富士通オートメーション(株)	(株)ゼットエムピー
身長		100cm	50cm	39cm
体重		20kg	約7kg	2.5kg
関節自由度	首	3自由度(目1，首2)	2自由度	
	腕(手を含む)	10自由度(5×2)	10自由度(5×2)	2自由度(1×2)
	腰	2自由度	1自由度	
	脚	12自由度(6×2)	12自由度(6×2)	12自由度(6×2)
	合　計	27自由度	25自由度	14自由度
サーボ・モータ		DCサーボ・モータ27個	RC用サーボ・モータ　4個(手開閉，首回転，頭うつむき/上向き)，他DCブラシレス・サーボ・モータ×21個	
販売価格/レンタル価格/非売		約12,000,000円	オープン価格(約5,700,000円)	約3,150,000円(開発モデル)

仕様 ＼ ロボット名		ロボット研究用プラットホーム(300万円以下)			ロボット機構部品セット
		FREEDOM	e-nuvo(イーぬーボー)	Robovie-M(ロボビー・エム)	YDH-PDS
開発主体		(株)ベストテクノジー	(株)ゼットエムピー	(株)国際電気通信基礎技術研究所(ATR)/ヴィストン(株)	(株)イトーレイネツ
身長		41cm	30cm	29cm	41cm
体重		2.5kg	1.35kg	1.9kg	2.5kg
関節自由度	首	1自由度			1自由度
	腕(手を含む)	8自由度(4×2)		8自由度(4×2)	8自由度(4×2)
	腰	2自由度		2自由度	
	脚	12自由度(6×2)	12自由度(6×2)	12自由度(6×2)	12自由度(6×2)
	合　計	23自由度	12自由度	22自由度	21自由度
サーボ・モータ		近藤科学 KRS-2345ICS/PDS-947FET	DCギアド・モータ 定格トルク390mN･m	SANWA ERG-VB×14 SANWA SPEC-AP2×8	近藤科学 KRS-2345ICS/PDS-947FET
販売価格/レンタル価格/非売		約500,000円	約680,400円	約398,000円	約138,000円(機構部のみ)

研究用プラットホームとして販売されているロボットは，技術情報が開示されています．制御アルゴリズムやプログラミングの研究が主目的の場合は，ロボット・メカの組み立てや制御ハードウェアに時間をかけるより研究用プラットホームを購入するほうが早道です．

市販機構部品を購入してロボットを組み立てる

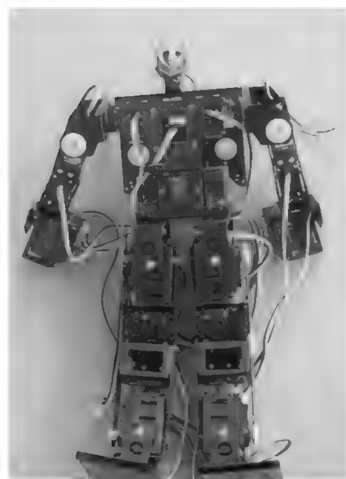
YDH-PDSはイトーレイネツが昨年末発表した二足歩行ロボット機構部品セットです．イトーレイネツはRCサーボ・モータ用ブラケットやオプション部品をセットにして商品化，二足歩行ブームの先導役を果たした会社です．YDH-PDSはその集大成ともいえる商品です．

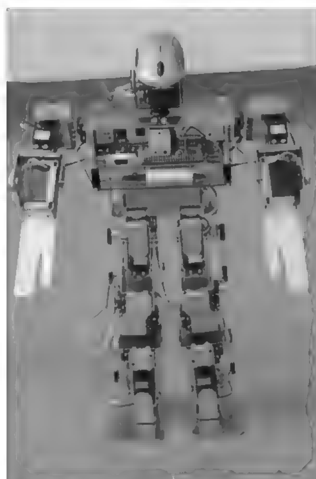
昨年末，一式入手して組み上げてみました．写真1はその外観です．サーボ・モータは近藤科学のKRS-2346ICS(トルク20kg/cm)を使いました．

YDH-PDSはベストテクノロジーと姫路ソフトワークスの制御基板に対応しています．第2章で紹介したROBO-01A基板はスペースと取り付け穴位置が合いません．新規開発のROBO-02はサイズをこのYDH-PDSに合わせましたが，写真撮影の段階では調整が終わっていませんでした．

二足歩行ロボット製作を極めていくと…機構部品の自作・外注は避けて通れない

研究用プラットホームや市販機構部品を使えばロボット開発の第一歩はクリアできます．しかし開発を進めていくと機構部品の製作が必要になってきます．イトーレイネツのオプション部品やYDH-PDSは最低限必要な部品しか含まれていません．頭部や手の機構部品は含まれていないので自作せざるを得ません．また大きなトルクがかかるひざ関節や足はロボットの安定性を左右するので，独自に開発した部品が必要になってきます．


写真1
YDH-PDSで組み上げたロボット


写真2
試作したロボットWSGH-1

ロボットの競技大会ROBO-ONEでは市販部品だけでは勝負にならないようでした．

ロボットの機構部品やアクチュエータは性能を左右する重要な要素技術です．本格的なロボット開発を行うためには，機構部品の開発は避けて通れません．

自動車や産業用機器製造メーカの場合は，商品開発のために機構部品の試作ラインをもっています．その気になればロボット部品の製作など朝飯前です．

大学や官公庁研究機関の場合には外注せざるを得ません．しかし，どこの業者に，どんな形で依頼すればよいのか，地域の実情に合わせて協力工場を見つける必要があります．

ロボット開発は，量産計画のない少量多品種生産になるので，コストも膨らんできます．たいへん厄介な課題ですが，早い時期に目処をつけておく必要があります．機構部品調達の手立てのないロボット開発プランは絵に描いたモチです．

自作・外注の第一歩は3次元CADで部品の設計

今回試作したWSGH-1(写真2)は，地域のロボット・イベントでデモ演技を行うとたいへん好評です．このロボットの頭部にはマウスを流用し，手は糸ノコとヤスリで仕上げた手作り部品です．糸ノコ，万力，ヤスリ，電気ドリルによる金属工作は，自作派の世界では数十年の歴史をもつ伝統工芸です．

しかし，同じ部品を10個作ろうとすると10倍の時間がかかります．また，ロボットの部品は精度を必要としますが，素人の金属工作では望むべくもありません．

ある金型業者に部品外注の留意事項を聞いてみました．この業者は中小企業ですが，自動車，電気メーカ，産業用機器メーカの金型を幅広く手がけています．金型メーカは1年に数えるほどしか使わない加工機も含めて，豊富な加工機をラインナップしています．この環境を生かしてロボット部品の試作も手がけています．

つい最近も年末年始の2か月ほどで，図面数十枚，部品点数1,000点を超える医療系ロボットの試作をこなしました．この業者にロボット機構部品を外注するコツを聞いてみました．

開口一番いわく，「3次元CADを使いこなして設計データで外注すればコストも納期も圧縮できる」とのことでした．現在，彼の工場はCAD/CAMによる自動機による加工が標準になっているからです．

部品試作は基本的に少量多品種生産で高コストです．「こんな部品が欲しいんですけど」という口頭や手書き図面の注文では，3D-CADに入力する作業が必要です．この過程で注文部品の詳細について打ち合わせをしたり，再検討をしたりといった，このコストも部品原価に入ってきます．

製作する部品の形状は注文主が設計します．メカの専門家で

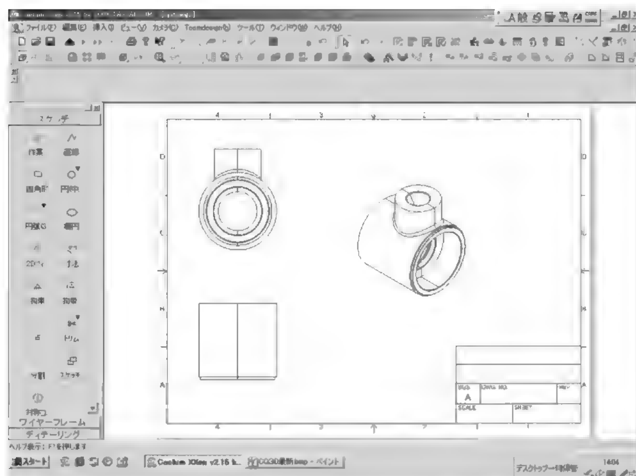


図1　試作したい部品を3D-CADに入力する(図版提供：トヨタケーラム社)

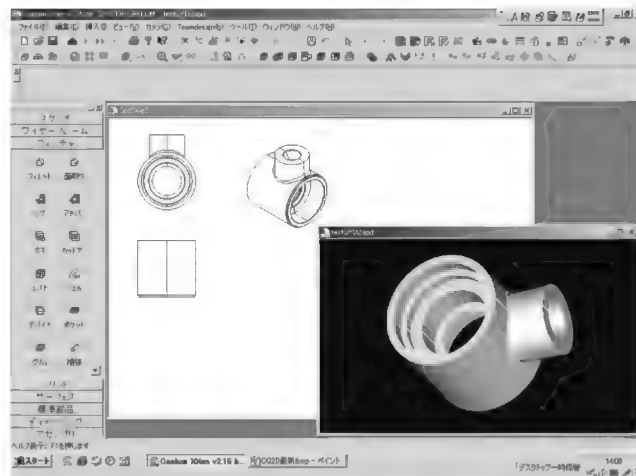


図2　設計した部品を3D表示で確認する(図版提供：トヨタケーラム社)

ないケースも多いと思いますが，部品設計の基本は押さえておく必要があります．この業者の口癖は「初心者の設計ミスで多いのは加工できない部品，組み立て不可能な部品が多い」というものです．

ドリルの刃先が入らないところに取り付け穴がある部品，これは製作不可能な部品です．ドライバが入らないところにタップが切ってある部品，これは組み立て不可能な部品です．業者によっては組み立て不可能な部品でもそのまま製作してしまいます．発注図面どおりの部品を製作すれば，業者としては責任をまっとうしたことになるからです．でき上がった部品を組み立てる段階で「あれ！」と気づいてもあとの祭り，発注者は高い授業料を払う羽目になります．

部品加工は板金，フライスによる削り出し，旋盤加工，ドリル加工，鋳造といろいろな手段があります．注文主は加工の専門家ではありませんが，製作工程をある程度念頭において設計する必要があります．

3D-CAD設計事始め

図1は3D-CADに入力した試作ロボットの部品です．図2はこれを3次元表示した画面です．3次元表示画像は角度を変えて最終完成部品の形状をチェックすることができます．

写真3は，アルミの塊りから削り出しで製作した部品に必要な穴を開けているところです．CAD/CAMを駆使した自動製作でも，実際の製作は複数の加工機を使っていくつかの工程を経て行われます．

CADデータになった設計図面があれば，不具合があっても修正は容易です．図1，図2は数少ない国産CADメーカであるトヨタケーラム社のCAD図面です．筆者の周りにはAUTOCADを使って設計を始めている技術者もいます．

3次元CADはフルセットの製品は数十万円から数百万円し



写真3　設計した3DデータでCAMを使って自動機により削り出して製作したのち，ドリル穴やタップは別工程で加工

ます．いきなり本製品を購入しても使いこなせないとむだになりますが，各社「お試し版」,「教育版」を用意しています．まずはこのあたりから始めてみるのが無難です．

アクチュエータ(モータ，ギア，サーボ回路)の設計…いちばん入手が容易なRCサーボ・モータ

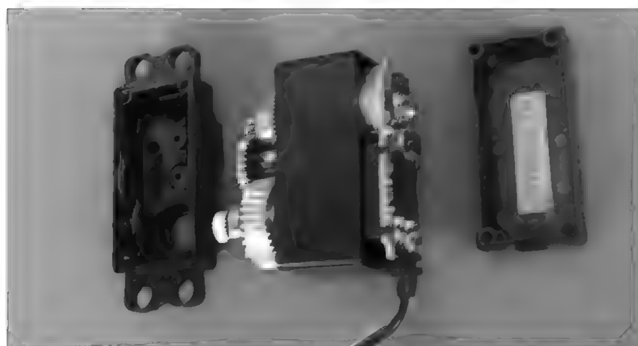
二足歩行ロボットの開発ではアクチュエータ(モータなど可動機構)の設計もたいへん重要です．据付型の産業用ロボットでは，アクチュエータ自体の重さはそれほど重要ではありません．しかし二足歩行ロボットの場合，軽くて強いトルクのアクチュエータは何よりも重要です．モータ，ギア(減速機構)，サーボ回路の重量はロボット自体の荷重負荷となるからです．

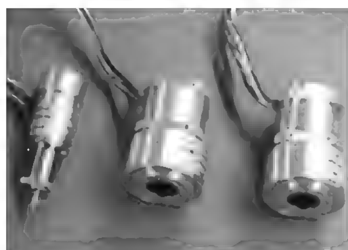
いちばん手軽で入手が容易なアクチュエータはRC(ラジコン)サーボ・モータです．これはラジコンの制御アクチュエータとして開発された部品です．制御信号を受けて模型自動車の

ステアリング，エンジン・バルブ，模型船の舵などを動かすために開発された部品です．二足歩行ロボットにRCサーボ・モータを使うことを最初に思いついた人には敬意を払わざるを得ません．

RCサーボ・モータはモータ，ギア，サーボ回路をモジュール化した商品です．写真4は近藤科学のPDS-2144FETを分解したパーツです．小さなスペースに，モータ，ギア，サーボ回路がぎっしり詰まっています．

1個1万円前後，ロボット1台に最低20個は必要ですから，高価な買い物です．しかしロボット部品としてはコスト・パフォーマンスの良いアクチュエータです．トルクが20kg/cmにおよぶものも出てきました．また一般の模型店で販売されている商品ですから，全国どこでも入手可能です．


写真4　サーボ・モータPDS2144FETの内部（ギヤ・ボックスと制御基板が組み込まれている）


写真5
HOAP-2に使われている
スマート・ギアド・モータ

中型ロボットのアクチュエータ

写真5は富士通オートメーションのHOAP-2に使われているサーボ・モータです．モータ，ギア，制御回路をまとめてスマート・ギアド・モータと呼ばれています．

モータは表2に示すTYPE-1，TYPE-2，TYPE-3の3種類が使い分けられています．トルクは市販のRCサーボ・モータと変わりません．しかし，ブラシレス・モータの採用により一般のブラシ付きモータ製品とくらべて長寿命です．

写真6，表3はモータ制御基板です．37×45mmの小さな基板にブラシレス・モータ・サーボ回路とUSBインターフェースが実装されています．図3はスマート・ギアド・モータを使ったサーボ系のシステム構成図です．

科学技術振興事業団が開催したロボットの展示会で担当者にモータと制御基板の外販について聞いてみました．すると「希望があれば外販も考えますが特注部品でけっこう高い値段になります」という回答が返ってきました．値段は20万円くらいになるかな…という感触でした．RCサーボ・モータのコスト・パフォーマンスの良さがわかります．

ASIMO，KOZOH Ⅲに使われている高級減速機構…ハーモニック・ドライブ

ロボットのアクチュエータは高度なノウハウが凝縮されてい

表2[1]　HOAP-2に使われているスマート・ギアド・モータの仕様

仕　様	Type-1	Type-2	Type-3
サイズ（mm）	φ22.0×53.4	φ35.0×52.5	φ35.0×56.5
重量（g）	60	140	150
モータ形式	ブラシレスDCモータ	ブラシレスDCモータ	ブラシレスDCモータ
位置センサ	入力軸2相エンコーダ	入力軸2相エンコーダ	入力軸2相エンコーダ
減速比	1/144	1/171	1/171
定格出力（W）	0.39	4.5	6
定格トルク（kgf-cm）	1	15	20
定格回転数（rps）	0.6	0.5	0.5
定格電流（A）	0.5以下	1以下	1以下
最大トルク（kgf-cm）	4	30	45
最大トルク時回転数（rps）	0.35	0.1	0.1
最大トルク時電流（A）	0.8以下	2以下	2以下
出力軸分解能（パルス/度）	176（4逓倍時）	209	209
トルク定数（kgf-cm/A）	10.6	19	26.8
無負荷回転数（rps）@24V	2.46	1.05	0.85
起動電流（A）	1.2	2.5	3.6
出力軸バック・ラッシュ（度）	0.99	0.8	0.8
出力軸剛性（Nm/度）	0.57	0.57	0.57

- ギアに特殊樹脂を採用しており非常に軽量（Type-2およびType-3）．
- ブラシレスDCモータの採用によりモータ部はメンテナンス・フリー．
- モータ・マグネットには強力なネオジウム・マグネットを採用．
- 通信インターフェースにUSBを採用しており，パソコンから容易に制御可能．
- USBハブを用いて複数のモータをネットワーク制御することが容易（最大25個を1ms周期）．
- モータ制御基板にはワンチップ・マイコンを搭載して，ローカルなインテリジェント制御を実現．
- モータ内部に回転検出2相エンコーダを搭載．
- フラッシュ・メモリに格納されたモータ制御ファームウェアは，USB経由で書き換えが可能．
- モータと基板を分離し，配置を変更することも可能．

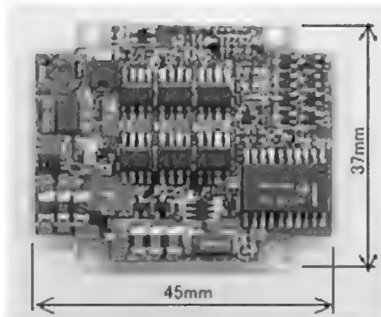


写真6　HOAP-2の
モータ制御基板

表3[1]　HOAP-2に使われているモータ制御基板の仕様

マイコン動作周波数16MHz
256Kバイト ROM
16Kバイト RAM
通信インターフェース　USB(12Mbps)
ユーザ用予備　I/O　3ビット
モータ用供給電源　24V
ロジック用供給電源　3.3V
USB経由で内蔵プログラム・オンライン書き換え可

るためか，なかなか技術情報が伝わってきません．そのような中である減速機構メーカのパンフレットに「ASIMOにも使われているハーモニック・ドライブ」という記述を見つけました．

ギアはモータの高速回転を減速し，ロボットに必要な高トルク駆動力を作り出す重要なメカ部品です．通常のギアだと1段で1:100の減速比は出せません．数段の減速ギア・ボックスを使って徐々に減速して行きます．

ところがハーモニック・ドライブは1段で1:30～1:320の減速比が得られる画期的な減速機構です．アメリカのマッサー氏が考案した原理をハーモニック・ドライブ・システムズ社が商品化しました．

ロボットの減速機構として必要な，

1) 小型・軽量
2) 大きなトルク容量
3) 高い減速比
4) 小さいバック・ラッシュ
5) 高精度
6) 部品数が少なく組み込みが容易
7) 優れた効率
8) 静かな運転

という特徴を備えています．

ロボットの減速機構としては理想的な部品ですが，唯一残念なのは高価であることです．ASIMO，KOZOH IIIにはこのハーモニック・ドライブが使われています．図4はハーモニック・ドライブの動作原理図です．

参考・引用＊文献
(1)＊富士通オートメーション，HOAP-2仕様書
(2)＊ハーモニック・ドライブ・システムズ，ハーモニック・ドライブ技術資料

よしだ・こうさく

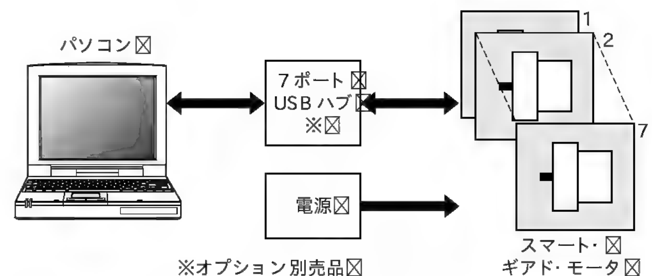


図3　スマート・ギアド・モータのシステム構成
(スマート・ギアド・モータは，USBインターフェースを備えたサーボ・モータ・モジュール)

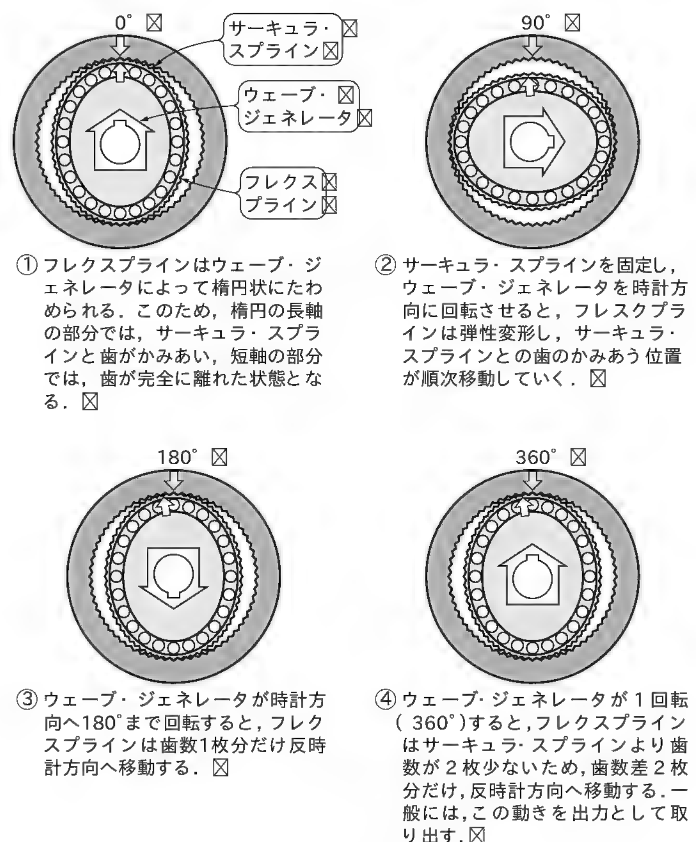


図4[2]　ハーモニック・ドライブの動作原理

Appendix 3

ロボットの機構とトルク設計



吉田 幸作

● トルクとは

ロボットの設計には，トルクということばが頻繁に出てきます．「近藤科学のKRSはトルクが20kg・cm」，「バッテリ電圧が下がってくるとモータのトルクが足りなくなってロボットが不安定になる」などです．

トルクとは回転力の大きさです（図A）．半径5cmと10cmの滑車に二つの力が加わっています．これらは互いに反対方向に回そうとしています．この力くらべ…はたして滑車はどちらの方向に回転するでしょうか．この答えを出すためには，

トルク（回転力）＝〔力の大きさ〕×〔中心からの距離〕

により，トルクの大きさを計算する必要があります．

時計まわりの回転トルク　＝ 12kg × 10cm
＝ 120kg・cm

反時計まわりの回転トルク　＝ 15kg × 5cm
＝ 75kg・cm

となるので，滑車は時計まわりに回転を始めます．

なお，この図では時計まわりの力を〔12kg〕と表記していますが，正確には〔12kgf〕とすべきです．〔重さ〕12kgの地上の物体に加わる〔重力の大きさ〕が12kgfです．ここでは一般の慣例に習って〔重さ〕も〔力の大きさ〕もkgで表記することにします．

● ロボットのサーボ・モータ…必要なトルクは？

ロボットの機構および駆動系の設計では，トルク計算は避けて通れません．図Bに示すロボットの足首関節サーボ・モータにかかるトルクを考察してみましょう．

ロボットの重量は3kg，ロボットの重心から足首サーボ・モータの駆動軸までの距離を20cmとします．ロボットに加わる重力は3kgfです．ロボットが直立姿勢の状態では力は垂直方向に加わるので，足首サーボ・モータのトルク負荷は0です．

ロボットが5°傾いたときのトルクはどうでしょうか．ロボットの荷重3kgの力は垂直方向と回転方向の成分に分解できます．足首サーボ・モータの回転軸に加わるトルクは，次のようになります．

トルク成分＝ 3kg × 20cm × sin（5°）
＝ 5.23kg・cm

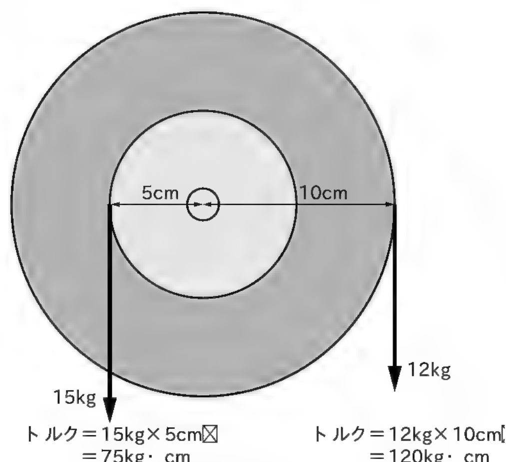
ちなみに10°傾くとトルクは10.42kg・cmとなります．

これは静止トルクの考察です．ロボットが何らかの理由でユラーッと傾いたときにぐっと持ちこたえて姿勢を正すために必要なトルクは，この2倍以上必要と考えられます．

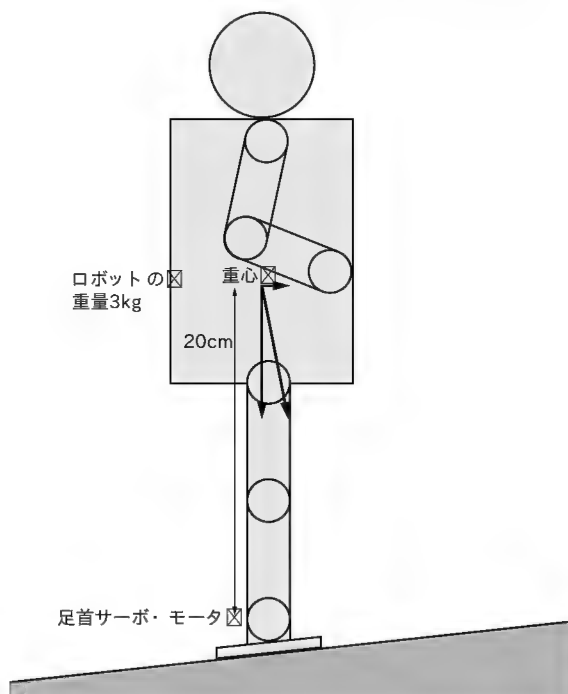
近藤科学のRCサーボ・モータ PDS-2144FET のトルクは13kg・cm，KRS-2346ICSは20kg・cm，外形はほとんど違いませんが，使ってみると違いは歴然です．

サーボ・モータは限界に近いトルクがかかるとブルブルと振動します．そして限界を超えるトルクがかかると，モータが脱調したり，焼き切れたり，内部ギヤ・ボックスの歯車が破損します．

図Bの機構ではサーボ・モータの駆動軸にロボットの荷重が加わりますが，これも機構設計の観点からすると好ましくないようです．「ロボットの荷重を支える支持軸と駆動軸は分離すべき」というのが知り合いの金型業者の意見です．



図A　トルクとは…回転力の大きさ×中心からの距離

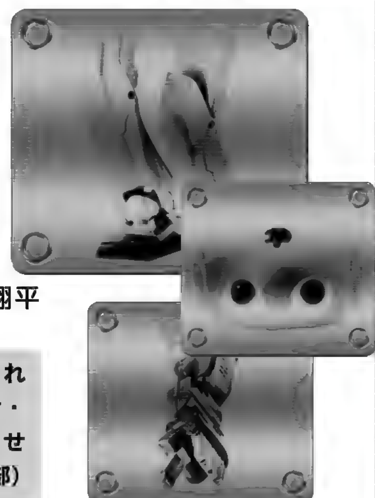


図B　ロボットにかかるトルクの計算

5. 自律歩行させるためのプログラム開発

二足歩行ロボットの制御アルゴリズムとプログラミング

吉田 幸作/岩田 正雄/藤田 正昭/
内海 裕憲/川畑 茂/長谷川 翔平



ロボットに二足歩行させるためのアルゴリズムの研究は盛んに行われている．そこで導き出された方法として，次のようなものがある．順動力学計算法，実際の人間の歩行動作をモーション・キャプチャする方法，逆運動学的計算による手法．ここでは，逆運動力学的手法によって実現させる手法を解説する．（編集部）

二足歩行ロボット制御の理論的研究と実践的アプローチ

二足歩行ロボットにおいて，最大の開発要素は，制御アルゴリズムとプログラミングです．そのため，実際のコーディングに至る前の制御アルゴリズムの開発が大きな課題となります．

人間は社会的に二足歩行をマスタします．二足歩行ロボットを歩かせるためには，左右の足を交互に出せばよい…．そのとおりですが，これがなかなか難しいものです．

二足歩行ロボットの自由度は下半身だけでも最低12自由度あります．腕，腰，頭部の自由度を加えると20以上になります．写真1は人間をフィギュアで表現したものです．人間は，

1) 左足に体重をかけ，つま先に力を移しながら
2) 右足を前に出し
3) 右足裏の角度を床の角度に合わせ

という，一連の関節制御を無意識のうちに行っています．

実際にこのポーズをロボットにとらせてみるとわかるのですが，左右の腕の振りや上半身の姿勢も人体の歩行動作中のバランスと密接に関連しています．人間が意識せずに行っている歩行動作をロボットに行わせるために，長年にわたって機械工学分野などで研究が進められてきました．

ジャイロを使って姿勢制御を行えば，ロボットを単に倒れないように制御するのは簡単ですが，二足歩行は姿勢制御だけの問題ではないことがわかります．

ロボットの歩行アルゴリズムの解明には，理論的アプローチと経験則から展開していく実践的アプローチがあります．学問として展開していくためには理論的アプローチは不可欠ですが，理論的な計算だけではロボットが歩行するために必要なデータを算出できないのも事実です．

ロボットの理論的実績を積んだ大学や研究機関のロボットと，実践的なノウハウの積み上げから生まれたロボットがROBO-ONEの試合で対戦した場合，理論的に優れたロボットが必ず勝利するとはいい切れないのがこの世界です．もちろん，理論的な解明なくして技術の進歩はありえないので，ロボットの歩行メカニズムの研究は重要です．しかし完成度の高いロボットは，理論的なバック・グラウンドだけでなく，実践的な経験から導き出された多くのノウハウによってはじめて可能となります．

サーボ特性のバラつき，ロボット・メカのオフセット補正，足の形状，床のすべり度…このようなたいへん泥臭い要素もロボットの安定した歩行動作に大きな影響を与えます．

二足歩行ロボットの研究は，歴史が浅いのでこのような理論的解明の問題と実践的なノウハウのバランスがクローズアップされていますが，よく考えるとこれは工学全般にいえることかもしれません．

ロボットを歩かせるためには足の関節角度を時々刻々と変化させる必要がありますが，この関節角度の時系列データを算出するためにいろいろな手法が研究されてきました．

ロボットの歩行アルゴリズムの理論的研究は，大きく分けて次の3グループに分類できます．それは，

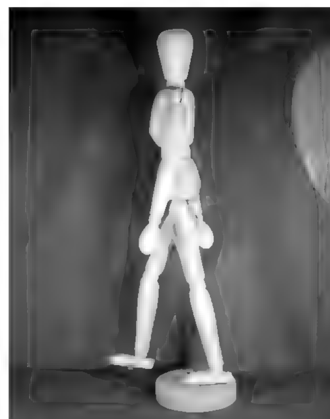


写真1　人間のフィギュア

1) 順動力学計算法
2) 実際の人間の歩行動作をモーション・キャプチャする方法
3) 逆運動学的計算による手法
です.

● 順動力学的手法

ロボットの足は各構成部品が関節でリンクした物体なので，構成部品の質量分布データと関節角度を与えることにより，歩行のシミュレーションが行えるはずです.

こういったことを「順動力学」などといいます．何だか特殊な学問のように聞こえますが，ロボットを構成する各部品の寸法と各関節のリンク角度から足の動きをシミュレーションする手法です．各部品の重量分布とリンク長を入力して運動方程式を用いて計算すれば，倒れない歩行アルゴリズムを導き出せるはずです.

このアプローチは理論的には正しい手法ですが，実際にコンピュータを使ってシミュレーションを行う段階で大きな壁が現れます．それは関節の自由度が多いため，シミュレーションに必要な計算量が爆発的に多くなり，現実的な時間内に必要な動作アルゴリズムを導き出せないという問題です.

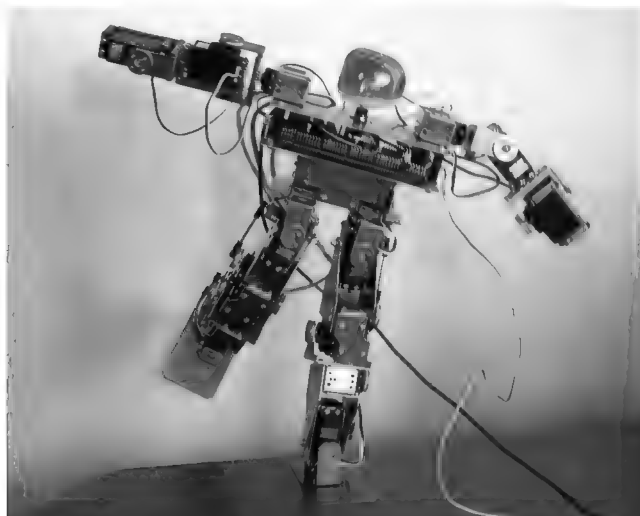


写真2　片足立ちのポーズ



写真3　片足立ちのときの足下

機械工学分野では，現在もこの計算量を短縮するためのアルゴリズムの研究が進められています．しかし，現状はまだ研究段階で，実際のロボットの歩行データを算出するツールとして使える段階にはないようです.

● 人間の歩行動作をモーション・キャプチャ

工学の歴史は現実モデルの模倣から始まりました．実際の人間の歩行動作をカメラを使ってモーション・キャプチャし，そのデータを解析して歩行アルゴリズムを導き出そうというのがこのアプローチです.

確かにロボットの関節リンク構造は，人間の関節構造を模倣して作られています．しかし完全に同一ではありません．まず構成物質からして異なるうえ，駆動メカニズムや各部分の質量分布なども異なります.

人間の動作はロボットの歩行アルゴリズムを導き出すヒントにはなります．しかし，現段階ではモーション・キャプチャしたデータから自動的に歩行データを導き出すことには成功していません.

モーション・キャプチャで得られたデータはそのままロボットの動作データとしては使えませんが，「力学フィルタ」と呼ばれる手法を使って動作データを導き出す試みは進んでいます．力学フィルタとは「力学的に実現不可能な運動データに必要な補正を加えて実現可能なデータに変換する」ツールです.

またROBO-ONEの大会でも見かけますが，「マスタ・スレーブ」操縦方式もモーション・キャプチャの応用といえるでしょう．これはロボットを操縦するオペレータの手足にセンサを付けて，オペレータの関節角度をロボットになぞらせる手法です.

● 逆運動学的手法による歩行アルゴリズム

二足歩行ロボットの理論的成果でもっとも実用に近いと考えられているのが，この「逆運動学計算手法」です．人間型ロボットの運動は手先，足先の位置および姿勢によって特徴付けられることが多いことに着目し，このデータから個々の関節の角度データを自動的に計算しようというのが，この逆運動学的手法です.

完全に一意的に個々の関節角度が決まるわけではありませんが，人間の手足の関節リンク構造の制約条件によって各関節の自由度を絞り込むことにより，かなり効率よく関節の角度データを生成することができます.

人間の歩行動作を解析して足の付け根と足先の位置を決めると，ほぼ自動的に各関節の屈曲角度は決まります．たとえば「右足先を10cm上げる」という動作を行わせるとき，「倒れないで」という条件を付けると，各関節の自由度はそれほど残っていません.

逆運動学的手法を使って「人間が歩くとき，足先はどのような軌跡をとるか…」ということから各関節の屈曲角度を決めていくと，闇雲に各関節角度を変えるよりもはるかに効率的に歩行に必要な各関節の角度データを算出することができます.

ロボットの重心制御…静止バランスの条件

● 二足歩行ロボットの力学

写真2は二足歩行ロボットの片足立ちのポーズです．二足歩行ロボットのパフォーマンス・ショーでは最高の見せ場です．写真3はこの片足立ちの足元をクローズアップした写真です．跳ね上げた右足と両手を使って絶妙なバランスをとるこの姿勢は，実際はロボット本体とモーション・デバッガを使えば比較的簡単に作れます．

このポーズをとるときには，静かに少しずつ動くので，運動の慣性をほとんど考える必要がありません．したがって，制御アルゴリズムとしては歩行動作よりも簡単です．

この片足立ちを少し力学的に見てみましょう．図1(a)は直立姿勢のロボットに加わる力を図示したものです．まずロボットの重心Pに重力Wが加わります．これに対して両足の裏に床半力P_R(右足)，P_L(左足)が加わります．ロボットが転倒しないで直立姿勢を保ち続けるためには，重力Wと床半力P_R，P_Lがバランスする必要があります．そのためには重力Wが床半力P_Rと床半力P_Lの合力に等しいだけでなく，力のモーメントもバランスする必要があります．

図1(b)は左足を上げた状態ですが，このときのバランスを考えてみましょう．重力Wと右足裏に加わる床半力P_Rは大きさが等しく方向が180°異なる力です．

ロボットが片足立ちを継続するためには，重力Wと床半力P_Rの力の向きが大きさが等しいだけでなく，一直線上になければなりません．もしわずかでも軸がずれていると，ロボットは右もしくは左方向に回転して転倒してしまいます．

片足立ちで静止したロボットの重力ベクトルWが床と交わる点を荷重点といいます．図1(b)のロボットの荷重点が床に接する足の裏の範囲内にある間はロボットは倒れません．

片足立ちのロボットが揺れると，ロボットの重心は図2(a)，(b)，(c)に示すように左右に移動します．それにつれて荷重点も図のように左右に移動しますが，足の裏が床面と平行な状態にあれば床反力の作用点も左右に移動します．その結果，重力ベクトルと床反力のベクトルは一直線上でバランスするのでロボットは倒れません．

ロボットの荷重点が足裏の輪郭外に飛び出すほど大きく揺れると，このバランスは崩れてロボットは倒れます．

また図2(a)'に示すように足の裏が床面と平行でない場合，床反力の作用点は足裏の右端に固定されます．この場合は荷重点が足裏の輪郭の範囲内にあってもロボットのモーメントはバランスしないので倒れます．

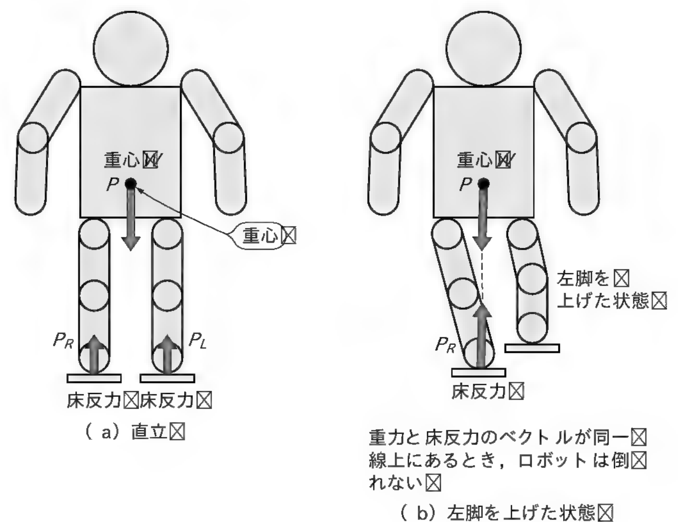


図1 二足歩行ロボットの力学

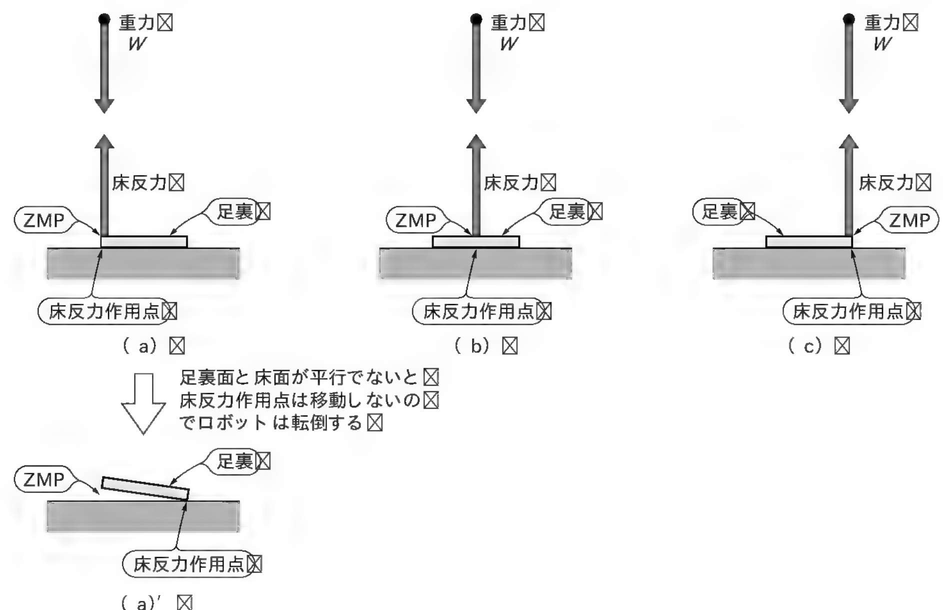


図2
静止ロボットは荷重点(ZMP:Zero Moment Point)が足裏の範囲内にあるときは安定する

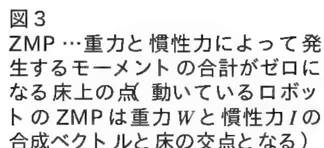


図3
ZMP…重力と慣性力によって発生するモーメントの合計がゼロになる床上の点（動いているロボットのZMPは重力Wと慣性力Iの合成ベクトルと床の交点となる）

● ZMP（Zero Moment Point）が足裏の輪郭線内にあるようにする

二足歩行ロボットをゆっくり静かに歩かせる場合は，重心制御の考えかただけで十分です．しかし，ある程度の速さで連続歩行を考える場合は重力のほかに慣性力も考慮する必要があります．

図3は慣性力を考慮した模式図です．ロボットの重力Wと慣性力Iの合成ベクトルが床面と交差する点をZMP（Zero Moment Point）と呼びます．連続的な歩行動作では，このZMPが足裏の輪郭線内にあることが安定歩行の条件になります．

二足歩行ロボット制御プログラム

リスト1は二足歩行ロボットの基本動作テスト・プログラム

リスト1　二足歩行ロボット制御プログラム

```
/*****************************************************************************
*****             二足歩行ロボット制御プログラム  robodemo2.C                *****
*****             original         M.Iwata    2003/11/03                    *****
*****************************************************************************/
#include <stdio.h>

#define  CMSTR   (*((unsigned short *)0xFFFF83D0))
#define  CMCSR0  (*((unsigned short *)0xFFFF83D2))
#define  CMCNT0  (*((unsigned short *)0xFFFF83D4))
#define  CMCOR0  (*((unsigned short *)0xFFFF83D6))

void set_pos(short[]);
void moveto(short[],int);
void step_ctrl(short []);
void move_by_key(short[]);
void wait(int);

unsigned short P_offset[]
                        ={480,377,630,555,535,570, /* 右足  下～上 R1～R6 */
                          420,467,565,495,530,540, /* 左足  下～上 L1～L6 */
                          512,195,940,890,         /* 右腕  下～上 r1～r4 */
                          512,910,135,160,         /* 左腕  下～上 l1～l4 */
                          690,90};                 /* 頭    下～上 h1～h2 */

/***** 初期位置データ *******/
short r0[]      ={0,  0,0,0,0,0,0,  0,0,0,0,0,0,  0,0,0,0,  0,0,0,0,  0,0};

/***** 直立補正データ *********/
short tyoku[]   ={1,  0,0,0,50,0,0,  0,0,0,-50,0,0,  0,0,0,0,  0,0,0,0,  0,0};

/***** 屈伸用データ *********/
short kussin1[] ={11,  0,0,0,50,0,0,  0,0,0,-50,0,0,  0,0,0,0,  0,0,0,0,0,  0,0};                       /*伸び*/
short kussin2[] ={12,  0,-185,-200,185,0,0,  0,185,200,-185,0,0,  0,0,0,0,  0,0,0,0,  0,0};             /*縮み*/
short kussin3[] ={13,  0,0,0,0,0,0,  0,0,0,0,0,0,  0,0,90,50,  0,0,-90,-50,0,  0,0};                    /*伸び，腕は後ろ*/
short kussin4[] ={14,  0,-155,-200,185,0,0,  0,190,200,-185,0,0,  0,0,90,50,  0,0,-90,-50,  0,0};       /*縮み，腕は後ろ*/

/***** 歩行用データ *********/
short wstep0[]  ={20,  102,-105,-20,-50,90,0,  125,85,-30,105,135,0,  0,0,0,0,  0,0,0,-50,  0,0};       /* 右足に重心 */
short wstep1[]  ={21,  102,-120,-40,-50,50,0,  165,240,325,-254,195,0,  0,0,0,-90,  0,0,0,0,  0,0};     /* 左足上げる */
short wstep2[]  ={22,  102,-120,-5,-100,45,0,  125,-40,80,-155,125,5,  0,0,0,-180,  0,0,0,0,  0,0};     /* 左足前へ */
short wstep3[]  ={23,  -10,-70,0,125,0,0,  5,-190,-10,-300,35,0,  0,0,0,0,  0,0,0,0,  0,0};             /* 左足着地 */
short wstep4[]  ={24,  -125,-75,30,-120,-135,0,  -87,105,20,50,-90,0,  0,0,0,50,  0,0,0,0,  0,0};       /* 左足に重心 */
short wstep5[]  ={25,  -115,-320,-325,254,-195,0,  -112,120,0,100,-50,0,  0,0,0,0,  0,0,0,90,  0,0};    /* 右足上げる */
short wstep6[]  ={26,  -115,-50,-140,145,-125,0,  -102,120,0,100,-45,0,  0,0,0,0,  0,0,0,180,  0,0};    /* 右足前へ */
short wstep7[]  ={27,  25,-120,-160,254,-35,0,  30,120,0,0,0,0,  0,0,0,0,  0,0,0,0,  0,0};              /* 右足着地 */

/***** 右足立ちデータ *********/
short migiash0[]={80,  102,-125,-20,0,95,0,  95,235,155,0,135,0,  0,0,0,0,  0,0,0,-50,  0,0};           /* 右足に重心 */
short migiash1[]={81,  102,-130,-170,275,-80,0,  135,395,448,-300,200,0,  0,0,-840,0,  0,0,760,0,  0,0}; /* 左足上げる*/
short migiash2[]={82,  102,-130,-170,275,-80,0,  135,165,213,-300,-20,0,  0,0,-840,0,  0,0,760,0,  0,0}; /* 左足ねじる*/
short migiash3[]={83,  57,-130,-170,275,20,0,  5,190,188,-300,-20,0,  0,0,-840,0,  0,0,760,0,  0,0};     /* 左足ねじる*/
short migiash9[]={89,  102,-125,-20,0,95,0,  95,235,155,0,135,0,  0,0,0,0,  0,0,0,-50,  0,0};           /* 右足に重心 */

short *last_r=r0;
int step_flg=0;

void main()
```

リスト 1　二足歩行ロボット 制御プログラム(つづき)

```
{
        char j,c,cnt,speed;

        printf("¥nHit key A(連続実行) or S(ステップ実行) ? ");
        do{     c=getchar();
                if(c=='S' || c=='s')    step_flg=1;
        } while(c!='a' && c!='A' && c!='S' && c!='s');
        if (step_flg!=1)        printf("¥n連続実行");
        else                    printf("¥nステップ実行");
        printf("します。¥n");

        set_pos(tyoku);                                 /* 直立 */
        speed=2;

        /* 屈伸2 */
        moveto(kussin3,speed);
        moveto(kussin4,speed);
        moveto(kussin3,speed);
        moveto(kussin4,speed);
        moveto(tyoku,speed);                            /* 直立 */

        /* 歩行 */
        for(cnt=0;cnt<4;cnt++){
                        moveto(wstep0,speed);           /* 右足に重心移動*/
                        moveto(wstep1,speed);           /* 左足を上げる */
                        moveto(wstep2,speed);           /* 左足を前へ */
                        moveto(wstep3,speed);           /* 左足着地 */
                        moveto(wstep4,speed);           /* 左足に重心移動 */
                        moveto(wstep5,speed);           /* 右足を上げる */
                        moveto(wstep6,speed);           /* 右足前へ */
                        moveto(wstep7,speed);           /* 右足着地 */
        }
        moveto(tyoku,speed);                            /* 直立 */

        /* 右足立ち */
        speed=1;
        moveto(migiash3,speed);
        moveto(migiash2,speed);
        moveto(migiash1,speed); wait(1000);
        moveto(migiash2,speed);
        moveto(migiash3,speed);
        moveto(tyoku,speed);                            /* 直立 */
}

/*****************************************************************************
*******************    set_pos(r)  r:Relative Position Value     **********
*****************************************************************************/
void set_pos(short r[])
{
        char k;
        unsigned short *pwm_adrs;
        pwm_adrs=(unsigned short *)0x00C00000;
        for(k=0;k<22;k++){
                *pwm_adrs++=r[k+1]+P_offset[k];
        }
        last_r=r;
    if(step_flg) step_ctrl(r);
}

/*******************************************************************************
**   補間関数  moveto(r2,speed)            r2:Relative Position Value      ****
**                                         speed:1~5                       ****
*******************************************************************************/
void moveto(short r2[],int speed)
{
        short j,k;
        short step;
        unsigned short *pwm_adrs,d;
        step=1000-speed*200+1;
        for(j=1;j<=step;j++){
                pwm_adrs=(unsigned short *)0x00C00000;
                for(k=0;k<22;k++){
                        d=(r2[k+1]-last_r[k+1])*j/step+last_r[k+1]+P_offset[k];
                        if(d>1023) d=1023;
                        if(d<0) d=0;
                        *pwm_adrs++=d;
                }
        }
        last_r=r2;
    if(step_flg) step_ctrl(r2);
}

/******************************************************************************
************************     現在位置表示          ****************************
```

リスト 1　二足歩行ロボット 制御プログラム(つづき)

```
*********************************************************************************/
void step_ctrl(short r[])
{
        char c,j;
        c=0;
        do{
                printf("¥n現在位置 "); printf("ID=%d  ",r[0]);
                for(j=1;j<=6;j++) printf("R%1d=%d  ",j,r[j]); printf("   ");
                for(j=1;j<=6;j++) printf("L%1d=%d  ",j,r[j+6]); printf("   ¥n");
                for(j=1;j<=4;j++) printf("r%1d=%d  ",j,r[j+12]); printf("   ");
                for(j=1;j<=4;j++) printf("l%1d=%d  ",j,r[j+16]); printf("   h1=%d   h2=%d",r[21],r[22]);
                printf("¥nスペースで続行、escで位置微調整モードに入ります。 ");
                while(1){
                        c=getchar();
                        if(c==' ') {printf("¥n"); return;}
                        if(c==0x1b)
                        {move_by_key(r); break; }
                }
        }while(1);
}
/*********************************************************************************
********************     位置微調整     r[] 現在位置データ(相対表記)            ******
*********************************************************************************/
void move_by_key(short r[])
{
        char rl,mno,pwm_no,c,d;
        unsigned short *pwm_adrs;
        pwm_adrs=(unsigned short *)0x00C00000;

        printf("¥n*** 微調整開始 (ESCで終了) ***¥n");
        rl=0;mno=1;pwm_no=0;
        do {
                        c=getchar();

                        if(c=='R') rl=0;
                        if(c=='L') rl=6;
                        if(c=='r') rl=12;
                        if(c=='l') rl=16;
                        if(c=='h') rl=20;
                        if(c>=0x31 && c<=0x36) {
                              mno=(c & 0x0f); pwm_no=mno-1+rl;
                              switch(rl){
                                case  0:printf("¥n右足");break;
                                case  6:printf("¥n左足");break;
                                case 12:printf("¥n右腕");break;
                                case 16:printf("¥n左腕");break;
                                case 20:printf("¥n頭");break;
                              }
                              printf("%d=%4d ",mno,r[pwm_no+1]);
                        }
                        if(c=='>') {
                             if(r[pwm_no+1]+5+P_offset[pwm_no]<1024) {
                                r[pwm_no+1]+=5;
                                *(pwm_adrs+pwm_no)=r[pwm_no+1]+P_offset[pwm_no];
                             }
                             printf("%4d ",r[pwm_no+1]);
                        }
                        if(c=='<') {
                             if(r[pwm_no+1]-5+P_offset[pwm_no]>=0) {
                                r[pwm_no+1]-=5;
                                *(pwm_adrs+pwm_no)=r[pwm_no+1]+P_offset[pwm_no];
                             }
                             printf("%4d ",r[pwm_no+1]);
                        }
        } while(c!=0x1b);
        printf("¥n*** 微調整終了 ***¥n");
        return ;
}

/*********************************************************************************
****************          Timer 0.001 s                                     *****
*********************************************************************************/
void wait(int w){
        int i;
        CMCOR0 = 875-1;
        CMCNT0 = 0;
        CMCSR0 = 0x0001;        // 28000000/32/875 = 1 mS
        CMSTR = 0x0001;         // CMT0 start
        for(i=w; i>0; i--){
                while((CMCSR0 & 0x0080)==0)
                        ;
                CMCSR0 = 0x0001;
        }
        CMSTR = 0x0000;         // CMT0 stop
```

です．このプログラムは第2章で紹介した二足歩行ロボット制御基板ROBO-01Aの上で走るプログラムです．

外部SRAM上で走らせてモーション・デバッグを行うプログラムですが，ステップ動作機能をはずせばSH7045Fの内蔵フラッシュ・メモリ上で走らせることもできます．

このプログラムには，

1) ロボットのサーボ初期値の設定
2) 直立，屈伸，歩行，片足立ちなどの基本動作を行わせるために必要なモーションの作成機能
3) キー操作によりロボット実機の各関節角度パラメータを変化させつつロボットのバランス状態を見てポーズを作成する「モーション・デバッガ」機能
4) 作成したモーション・データをつなぎ合わせてロボットの動きを作る補間機能

が組み込まれています．

二足歩行ロボットのサーボ設定値と制御角の関係…サーボ初期値の設定

二足歩行ロボットの基本姿勢は直立です．世間で言うところの「気を付け」の姿勢です．ロボットのサーボは初期値を設定したときに直立の姿勢になるように組み立てます．

リスト1の

```
unsigned short P_offset[]
={480,377,630,555,535,570,
                     /* 右足  下～上 R1～R6 */
  420,467,565,495,530,540,
                     /* 左足  下～上 L1～L6 */
  512,195,940,890,   /* 右腕  下～上 r1～r4 */
  512,910,135,160,   /* 左腕  下～上 l1～l4 */
  690,90};           /* 頭    下～上 h1～h2 */
```

は直立の姿勢をさせるための各関節サーボ設定値です．

このプログラムでは，第2章で紹介したROBO-01AのPWMコントローラPWM0～PWM21にロボットの関節サーボを次のように割り当てています．

PWM00～PWM05　右足の下（足首）から上へ順々に
PWM06～PWM11　左足の下（足首）から上へ順々に
PWM12～PWM15　右腕の下（手首）から上へ順々に
PWM16～PWM19　左腕の下（手首）から上へ順々に
PWM20～PWM21　首の関節（左右回転），（上下回転）

`short P_offset[]`の値は直立姿勢のパラメータですが，関節によって設定初期値が異なります．

図4は足の膝関節の例ですが，直立姿勢のとき関節は一直線になります．膝関節は図のように内側には曲がりますが，反対側にはほとんど曲がりません．

直立姿勢は関節自由度の端にあるので，図のように，

PWM設定値＝100

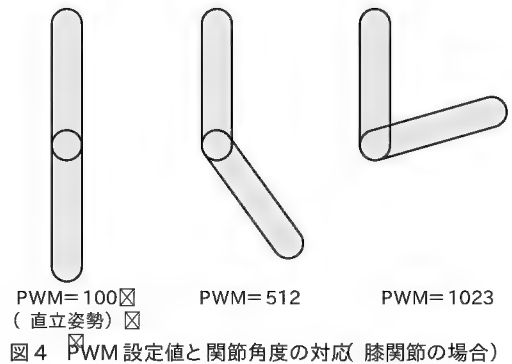


図4　PWM設定値と関節角度の対応（膝関節の場合）

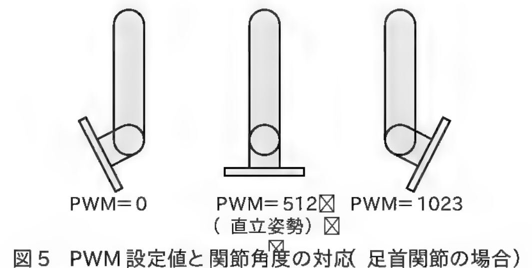


図5　PWM設定値と関節角度の対応（足首関節の場合）

のときに直立姿勢になるように組み立てます．0ではなくて100に設定するのは，微調整の余地を残すためです．

このような設定で組み立てると，PWM設定値を512，1023と変化させたときに，膝関節は図のように変化します．

図5は足首関節の例です．足首は直立姿勢の状態から左右に±90度の範囲で角度制御を行います．このように両方向に動作する関節のサーボは，

PWM設定値＝512

で直立姿勢になるように組み立てます．

リスト1の設定値`P_offset[]`は100，512などの理想値になっていません．これはロボットの機械的なオフセットを微調整した値です．

ロボットを組み上げるときは，各サーボに論理的な初期値

```
unsigned short P_offset[]
={512,512,100,512,512,
                     /* 右足  下～上 R1～R6 */
  512,512,900,512,512,
                     /* 左足  下～上 L1～L6 */
  512,100,900,900,   /* 右腕  下～上 r1～r4 */
  512,900,100,100,   /* 左腕  下～上 l1～l4 */
  512,100};          /* 頭    下～上 h1～h2 */
```

を設定した状態で直立姿勢になるように組み上げます．

しかし，実際に組み上げる過程で機械的なオフセットが必ず発生します．RCサーボの場合はサーボ・ホーンをねじ山の噛み合わせで挿入しますが，この過程でねじ山一つ分のオフセットが発生します．

機械的なオフセット値は，ロボット1台1台によって異なり

ます．これを，プログラムの初期値P_offset[]を調整してロボットが直立姿勢になるようにします．

二足歩行ロボット制御ことはじめ…サーボ位置制御プログラム(初期値設定)

それでは具体的な手順を説明します．ロボットの電源を入れてリスト1のプログラムを走らせると，

Hit key A(連続実行) or S(ステップ実行) ?

とメッセージが表示されるので，〔S〕をキー入力して〔ステップ実行〕を選択します．すると，

〔ステップ実行します.〕

というメッセージが表示されてロボットは直立姿勢をとります．しかしオフセット調整が済んでいない状態では，ロボットは多少崩れた姿勢をとります．プログラムは，

現在位置=…

と各サーボの現在値を表示します．そして，

スペースで続行，escで位置微調整モードに入ります.

と，〔ステップ実行〕と〔位置微調整モード〕の選択を求めてくるので，〔esc〕キーを押して位置微調整モードに入ります．

次にオフセット値を変更したいサーボを指定します．たとえば右足の足首を変更したい場合は，

R1

とキー入力すると現在値が表示されるので，キー入力

>　増加

<　減少

によりオフセット値を変更します．

オフセット値の変更と同時にロボットの姿勢も変わるので，これを見ながら最適なオフセット値になるようにします．

最適になったら次に変更したい関節を指示します．これをすべての関節について行うと，ロボットはきれいな直立姿勢になるはずです．

このオフセット値は電源を切ると消えてしまうので，メモを取ってプログラムのソースを書き直しておきます．

このオフセット値はロボットごとに異なるので，ロボットごとに別々のパラメータのソース・プログラムをもつ必要があります．複数のロボットを所有している場合は，ソースの管理が煩雑になります．後ほど紹介するパラメータ管理プログラムを追加すると，ロボットごとのオフセット値とモーション・データを基板上のフラッシュ・メモリに記録して管理することができます．

モーション・デバッガによるポーズ作成

オフセット値の設定が終了したら，次はロボットのモーションの制作に移ります．屈伸，歩行，片足立ちなど，ロボットの動作はいくつかのポーズをつなぎ合わせて作ります．

たとえば「歩行」動作は，

1) 右足に重心を移す
2) 左足を上げる
3) 左足を前に出す
4) 左足を着地する
5) 左足に重心を移す
6) 右足を上げる
7) 右足を前に出す
8) 右足を着地する

という八つのポーズに分解することができます．

リスト1のプログラムにはロボットの動作の基本となるポーズの関節データを制作する機能も組み込まれています．先ほどの直立オフセット値の設定と同じ要領で歩行に必要な八つのポーズの関節データを制作することができます．

リスト1の，

```
/***** 歩行用データ *********/
short wstep0[]
    ={20,  102,-105,-20,-50,90,0, ・・・
short wstep1[]
    ={21,  102,-120,-40,-50,50,0, ・・・
short wstep2[]
    ={22,  102,-120,-5,-100,45,0, ・・・
short wstep3[]
    ={23,  -10,-70,0,125,0,0,  ・・・
short wstep4[]
    ={24,  -125,-75,30,-120,-135,0, ・・・
short wstep5[]
    ={25,  -115,-320,-325,254,-195,0, ・・・
short wstep6[]
    ={26,  -115,-50,-140,145,-125,0, ・・・
short wstep7[]
    ={27,  25,-120,-160,254,-35,0 ・・・
```

はこの手順を経て制作したものです．配列の最初の変数はパラメータ管理用のIDナンバで，ロボットの制御には関与しません．

モーション・データは各ポーズごとにロボットのバランスを前提に順次制作していきます．リスト1のプログラムでは，電源を切るとモーション・データは消えるので，メモを取ってソースを書き換えます．

屈伸，歩行，右足立ちのデータができたら，今度は〔連続実行〕で動かしてみましょう．リスト1に組み込まれているモーション補間関数，

moveto(short r2[],int speed)

によりロボットは連続動作を開始します．

この歩行動作は安定したポーズ・データをつなぎ合わせたもので，運動の慣性などは考慮していないので「静歩行」と呼ばれています．速い歩行は無理ですが，とりあえず歩かせることはできるはずです．

連続動作で倒れるようであれば，各ポーズのモーション・データを手直しします．着地する足の角度や，各ポーズのバランスを手直しすればうまくいくはずです．

二足歩行ロボット制御プログラム

リスト 1のプログラムを連続モードで実行すると，

1）屈伸2回
2）4歩歩く
3）片足立ち

を順に行います．

このプログラムに使われている関数の概要を説明します．関数 `set_pos(short r[])`は関節データ `short r[]`を PWM コントローラにセットします．関節データは 22個の 1次元配列データです．

関数 `moveto(short r2[],int speed)`はモーションの補間を行います．ロボットの現在のポーズから配列 `short r2[]`で表現されるポーズへ線形補間を行います．

動作速度は引き数 `int speed`により 5段階の設定が可能です．このプログラムはモーション・データの作成を主目的に作られているので，補間ステップの刻みを変えることによって実行速度を調整しています．

ROM 化の段階では速度調節を正確なハードウェア・タイマを使った関数 `wait(int w)`を使って書き換える必要があります．

関数 `wait(int w)`は SH7045F の内蔵タイマを使って，

遅延時間（ms）＝w

を発生させる関数です．

`step_ctrl(short r[])`は，ロボットの関節データ（現状）をコンソールに表示する関数です．表示後この関数は〔ステップ実行の続行〕もしくは〔位置微調整〕を行います．

`move_by_key(short r[])`は実際にコンソール・キー入力により関節データの微調整を行うプログラムです．微調整を行う関節を指定した後，キー〔<〕,〔>〕によって増減処理を行います．

フラッシュ・メモリを活用したパラメータ管理プログラム

リスト 1のプログラムでは初期値およびモーション・データを確定した後，プログラムのソースを書き換えて保存する必要があります．実際にやってみるとこの作業はけっこう煩雑です．また，複数のロボットがある場合は，ロボットごとに異なるパラメータのソースを管理するのはたいへんです．

初期値およびモーション・データをロボット自体に記憶させれば，この煩雑さから解放されます．リスト 2は ROBO-01A の外部フラッシュ・メモリを活用したパラメータ管理機能を組み込んだプログラムです．

GUI プログラミング・ツール

リスト 1のプログラムは，モーション・デバッガ機能の搭載によって使い勝手が大幅に向上しました．次のステップとして

リスト 2　フラッシュ・メモリを活用したパラメータ管理プログラム（追加分のみ）

```
#define mark    0x0080FF80

void flash_write(short pose, short[]);
void flash_read(short pose, short[]);
void flash_erase(int _i);
void flash_save();
void flash_load();
void dump();


void main()
{
        char j,c,cnt,speed;

        if( *((short *) mark) == 0x5A5A )         flash_load();
                                                /* flash load */

        printf("¥nHit key A(連続実行)  or  S(ステップ実行)  ? ");
        do{     c=getchar();
                if(c=='S' || c=='s')    step_flg=1;
        } while(c!='a' && c!='A' && c!='S' && c!='s');
        if (step_flg!=1)        printf("¥n連続実行");
        else                    printf("¥nステップ実行");
        printf("します。¥n");

        set_pos(tyoku);                                 /* 直立 */
        speed=2;

        /* 屈伸2 */
        moveto(kussin3,speed);
        moveto(kussin4,speed);
        moveto(kussin3,speed);
        moveto(kussin4,speed);
        moveto(tyoku,speed);                    /* 直立 */

        /* 歩行 */
        for(cnt=0;cnt<4;cnt++){
                        moveto(wstep0,speed);   /* 右足に重心移動*/
                        moveto(wstep1,speed);   /* 左足を上げる */
                        moveto(wstep2,speed);   /* 左足を前へ */
                        moveto(wstep3,speed);   /* 左足着地 */
                        moveto(wstep4,speed);   /* 左足に重心移動 */
                        moveto(wstep5,speed);   /* 右足を上げる */
                        moveto(wstep6,speed);   /* 右足前へ */
                        moveto(wstep7,speed);   /* 右足着地 */
        }
        moveto(tyoku,speed);                    /* 直立 */

        /* 右足立ち */
        speed=1;
        moveto(migiash3,speed);
        moveto(migiash2,speed);
        moveto(migiash1,speed); wait(1000);
        moveto(migiash2,speed);
        moveto(migiash3,speed);
        moveto(tyoku,speed);                    /* 直立 */

        if(step_flg){
                printf("¥ndata save ? Y/N --> ");
                c=getchar() & 0x5F;
                if(c == 'Y' )    flash_save();

        }
```

リスト 2　フラッシュ・メモリを活用したパラメータ管理プログラム（追加分のみ）

```
//===============途中省略=====================

/****************************************************************
*******************   フラッシュ・メモリ　管理関数          ******
***************************************************************/
void dump()
{
        int     _i, _j, _k;

        for(_j=0; _j<11; _j++){
                for(_k=0; _k<23; _k++){
                        _i=0x00800000 + _j*128 + _k*2;
                        if(_k==_k/6*6)  printf("   ");
                        if(_k==_k/12*12)        printf("¥n");
                        printf("%6d",(*((short *) _i)));
                }
                printf("\n");
        }
        printf("¥n    %4X",(*((short *) mark)));
}

void flash_save(void)
{
        flash_erase(0);
        flash_write(0,P_offset);
        flash_write(1,r0);
        flash_write(2,kussin1);
        flash_write(3,kussin2);
        flash_write(4,kussin3);
        flash_write(5,kussin4);
        flash_write(6,wstep0);
        flash_write(7,wstep1);
        flash_write(8,wstep2);
        flash_write(9,wstep3);
        flash_write(10,wstep4);
        flash_write(11,wstep5);
        flash_write(12,wstep6);
        flash_write(13,wstep7);
        flash_write(14,migiash0);
        flash_write(15,migiash1);
        flash_write(16,migiash2);
        flash_write(17,migiash3);
        flash_write(18,migiash9);
        flash_write(511,marker);
}


void flash_load(void)
{
        flash_read(0,P_offset);
        flash_read(1,r0);
        flash_read(2,kussin1);
        flash_read(3,kussin2);
        flash_read(4,kussin3);
        flash_read(5,kussin4);
        flash_read(6,wstep0);
        flash_read(7,wstep1);
        flash_read(8,wstep2);
        flash_read(9,wstep3);
        flash_read(10,wstep4);
        flash_read(11,wstep5);
        flash_read(12,wstep6);
        flash_read(13,wstep7);
        flash_read(14,migiash0);
        flash_read(15,migiash1);
        flash_read(16,migiash2);
        flash_read(17,migiash3);
        flash_read(18,migiash9);
        flash_read(511,marker);
}

void flash_read(short pose, short r[])
{
        short _j;
        for(_j=0; _j<23; _j++){
                r[_j] = (*((short *) (0x00800000
                                                + pose*128 + _j*2) ));
        }
}

void flash_write(short pose, short r[])
{
unsigned int _i = 0x00800000 + ( pose<<7 ) ;
unsigned int _j;
unsigned char _a;
#define  K5      (*((unsigned char *)0x00800555))
#define  KA      (*((unsigned char *)0x008002AA))
        for(_j=0; _j<=23; _j++){
                K5 = 0xAA;
                KA = 0x55;
                K5 = 0xA0;
                _a = r[_j]>>8;
                *((unsigned char *) _i) = _a;
                while((*((unsigned char *) _i) ) != _a)
                        ;
                _i++;

                K5 = 0xAA;
                KA = 0x55;
                K5 = 0xA0;
                _a = r[_j];
                *((unsigned char *) _i) = _a;
                while((*((unsigned char *) _i) ) != _a)
                        ;
                _i++;
        }
}

void flash_erase(int _i)              // _i = erase sector No. */
{
#define  K5      (*((unsigned char *)0x00800555))
#define  KA      (*((unsigned char *)0x008002AA))
        K5 = 0xAA;
        KA = 0x55;
        K5 = 0x80;
        K5 = 0xAA;
        KA = 0x55;
        if(_i == 8){
                K5 = 0x10;
                while((K5 & 0X80) == 0x00)
                        ;
        }
        else{
                _i = ( _i<<16 ) + 0x00800000;
                *((unsigned char *) _i) = 0x30;
                while((*((unsigned char *) _i) & 0X80) == 0x00)
                        ;
        }
}
```

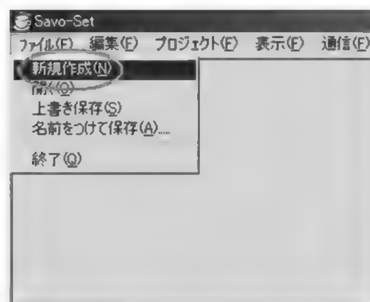


図 6
ファイル→新規作成

「C言語を使わないで二足歩行ロボット開発ができるプラットホームを作ってみよう」ということになりました．

二足歩行ロボットの開発は，メカ（機構）から電子回路，ソフトウェアの総合的な設計力が求められます．開発ターゲットをメカおよび電子回路におく場合は，C言語が使いこなせなくてもロボット開発ができるツールがあれば便利です．

筆者のチームでは2，3週間前からGUIロボット制御プログ

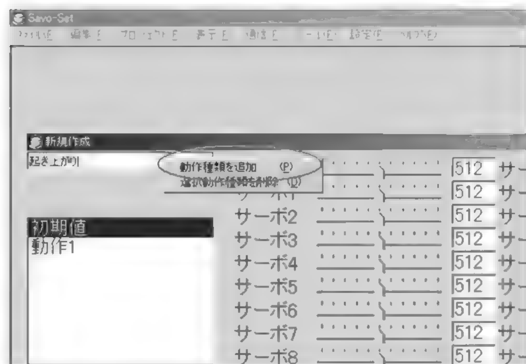


図7　動作種類を追加する

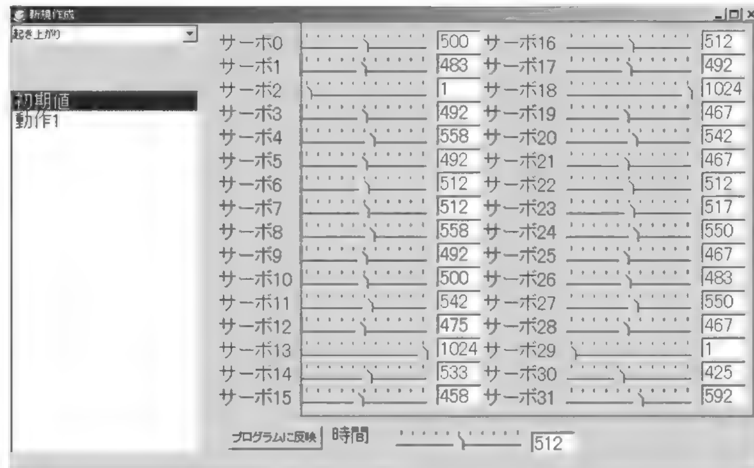


図8　動作1のサーボ角度情報を設定する



図9　ロボットの状態のグラフィックス表示

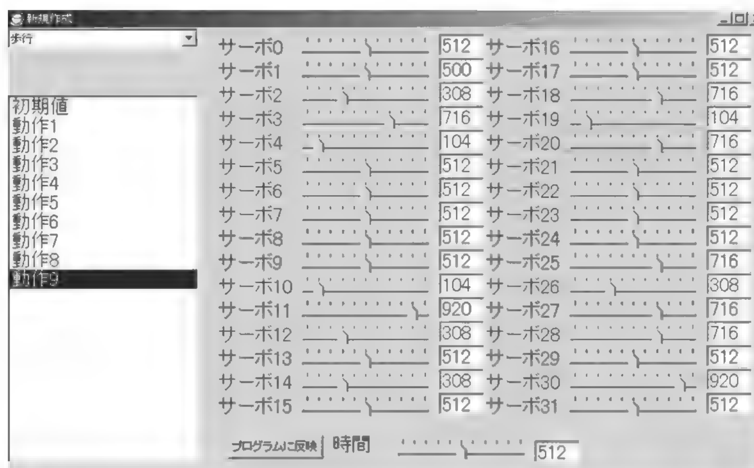


図11　順々に動作を追加していく

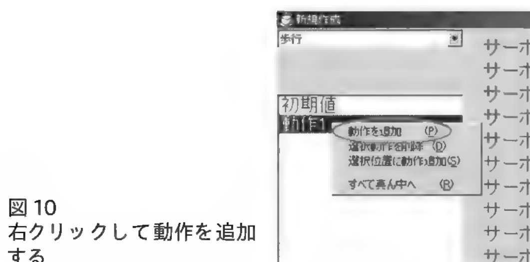


図10
右クリックして動作を追加する

ラミング・ツールの開発が始まりました．まだ開発途中ですが，その目標仕様を紹介します．

Windowsパソコンと ROBO-01A（もしくは ROBO-02）をシリアル・ケーブルで接続した状態でツール（Savo-Set）を起動します．そして図6に示すように

　　ファイル → 新規作成

をクリックします．すると図7の新規作成ウインドが開くので，左上のコンボ・ボックスで動作を選択します．

図7に示すようにすべてのサーボについて初期値を設定した後，図8に示すように動作1（ポーズ）のサーボ角度情報をセットします．このとき，操作画面には図9に示すようにロボットの状態をグラフィックスで表示します．それと同時にシリアル・ケーブルで接続されたロボット実機も動作するので，ロボットのバランス状態を見ながらパラメータを変えることができます．

さらに図10に示すように動作（ポーズ）を追加し，サーボの設定項目に数値を入力していきます．順々に動作（ポーズ）を追加していくと図11のように一連の動作が登録できます．

このようにして制作したモーション・データは画面上のグラフィックス表示ロボットおよび実機ロボットで動作を確認することができます．

画面上のロボットは主として関節角度とロボットのポーズ概要の確認用です．安定性の判定までは難しいと思います．目標仕様では図9に示すように，Cのソース・コードまで生成する予定です．

よしだ・こうさく/いわた・まさお/ふじた・まさあき/うつみ・ひろのり/かわばた・しげる/はせがわ・しょうへい

6. 加速度センサ，ジャイロ・センサ，感圧センサで検出

ロボットに使われるセンサ技術

吉田 幸作

人間が歩行するときには平衡感覚を保つために，加速度や角速度，足の裏の圧力などを感じ取って姿勢を保っている．これらの感覚を検出するセンサを紹介する．

（編集部）

二足歩行ロボットに使われるセンサ技術

表1は，プロローグで紹介したおもな二足歩行ロボットに使われているセンサの一覧表です．ロボットの歩行および姿勢制御に多く使われているのは，加速度センサ，ジャイロ・センサ，感圧センサです．

加速度センサおよびジャイロ・センサはロボットの姿勢制御，運動制御の高度化に役立っています．感圧センサはロボットの足の裏の4隅に貼って，床反力の分布を検出するのに大きな役割を果たしています．

CCDカメラを搭載しているロボットはかなり多くありますが，ロボットの運動制御用というよりは，情報収集用がほとんどのように感じられます．カメラからの画像をロボットの中で判断して自律動作に役立てる画像処理技術はまだ開発が始まったばかりです．CCDカメラの画像データをロボット外のホスト・コントローラに送ってパソコンやワークステーションで処理するのが精一杯です．

加速度センサを使って2次元の加速度を検出する

現在，加速度センサは国内外の10社以上から供給されています．ピエゾ抵抗素子を使った製品とMEMS（マイクロマシンニング）技術を使った静電容量型の製品があります．

その多くは3次元加速度センサですが，この分野でトップ・シェアの製品はアナログ・デバイセズ社の2次元センサです．図1はアナログ・デバイセズ社の最新製品ADXL311の機能ブロックです．

図2に示すように5×5mmのセラミック・リードレス・チップ・キャリアに入った，たいへん小さなデバイスです．増幅回路も内蔵されていて，加速度出力がアナログ値で得られます．

このデバイスは従来からあったADXL202Eのローコスト品です．ADXL202Eはアナログ出力とPWMディジタル出力の両方を備えています．

いずれもデータシート上の計測レンジは±2Gですが，実際は±3～4Gまで計測可能です．ADXL202Eの耐衝撃特性は500Gですが，ADXL311は3500Gまで保証されています．

加速度計測レンジが±10Gまで必要な用途にはADXL210E

表1 各センサが使われているロボットの数

加速度センサ	10
ジャイロ・センサ	9
力センサ	3
感圧センサ	6
測距センサ	3
温度センサ	4
タッチ・センサ	4
CCDカメラ	11

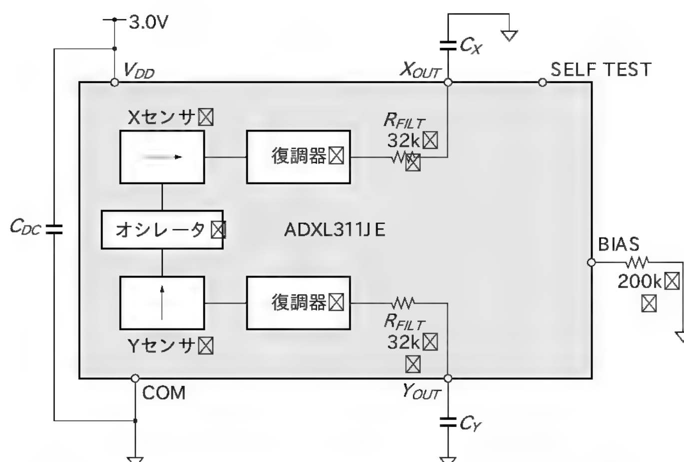
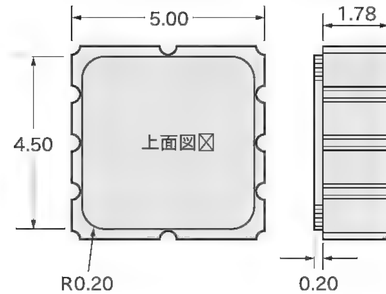


図1[1] アナログ出力加速度センサADXL311（アナログ・デバイセズ社）の機能ブロック図



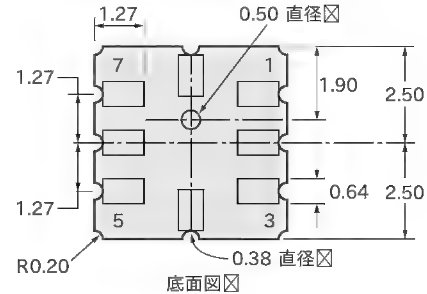


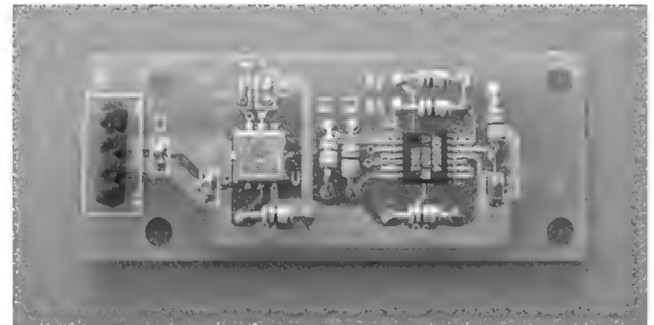
図2[1]
加速度センサADXL311のパッケージ
（5×5mmセラミック・リードレス・チップ・キャリア）

が用意されています．ADXL311，ADXL202E，ADXL210Eはパッケージとピン配列が共通仕様になっています．実装基板を共通にして用途に応じて実装チップを使い分けるという使いかたもできます．

ADXL311のアナログ出力は，図1の機能ブロック図に示すように*RC*フィルタからの出力です．X_{OUT}，Y_{OUT}端子をSH7045FなどのマイクロプロセッサA-Dコンバータに直結するとインピーダンス・ミスマッチによる計測誤差が発生します．

ADXL311をロボット部品として使うため，写真1に示す2軸加速度センサ基板を作りました．図3はその回路図です．加速度センサのアナログ出力を電圧フォロワ回路で受けて出力します．

ADXL311の特性は，データシートには電源電圧＝＋3.3Vの特性しか掲載されていません．しかし，このデバイスは＋5Vでも動作します．図3の基板は第2章で紹介したROBO-01AおよびROBO-02のアナログ入力端子に直結できます．


写真1　2軸加速度センサ基板

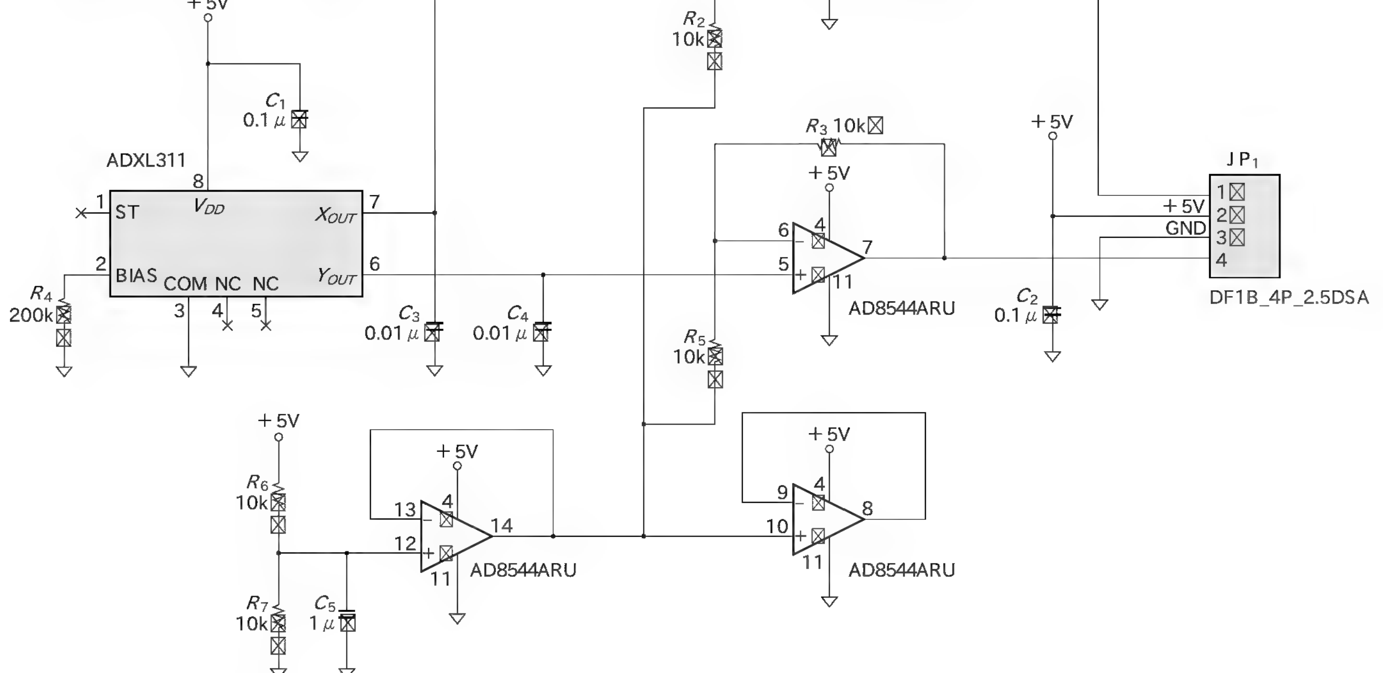


図3　2軸加速度センサ回路

高性能ジャイロ・センサADXRS300を使って角速度を検出する

加速度センサと並んでロボットの姿勢制御に大きな役割を果たしているのがジャイロ・センサです．ジャイロは図4に示すように回転の角速度を電圧値として取り出すことができるセンサです．

ジャイロ・センサにもピンからキリまであります．ビデオの手ぶれ防止に使われている圧電ジャイロは直線性が，± 5％FS（フルスケール）程度です．ところが航空機にも使われる光ファイバ・ジャイロは直線性が，± 0.1％FSに及びます．

光ファイバ・ジャイロは1個50万円もする部品ですが，なんとあのASIMOには3個も搭載されているという話を聞いたことがあります．ASIMOも技術的な改良が随時行われているので，現行機種に搭載されているかどうかは定かではありません．

図5は，アナログ・デバイセズ社のレート・ジャイロADXRS300の機能ブロック図です．

このジャイロ・チップの直線性は標準値で，± 0.1％と，光ファイバ・ジャイロに，肉薄する特性を備えています．

ADXRS300のパッケージは，図6に示す32ピン・チップ・スケールBGAです．BGAパッケージなので基板を開発しないと特性評価もできません．写真2は評価のために開発したジャイロ基板，図7は回路図です．

ADXRS300は図4に示したように，時計方向の回転に対して増加する角速度信号を出力します．

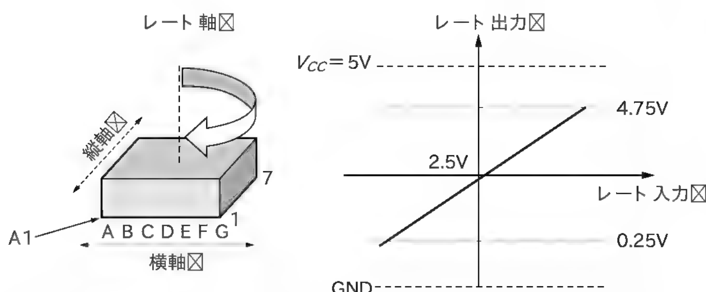


図4　ジャイロ・センサADXRS300のレート出力（時計方向回転でレート出力は増加する）

感圧ゴム・センサを使って姿勢を検出する

ロボットの歩行制御において，床反力の制御が重要な意味をもつことがわかってきました．ロボットが倒れる直前には足裏の特定部分に床反力が集中します．足裏の4隅に感圧センサを貼って絶えずモニタリングすることにより，ロボットが転倒するかどうかを早めに感知して姿勢の建て直し，もしくは受身体制をとらせることができます．

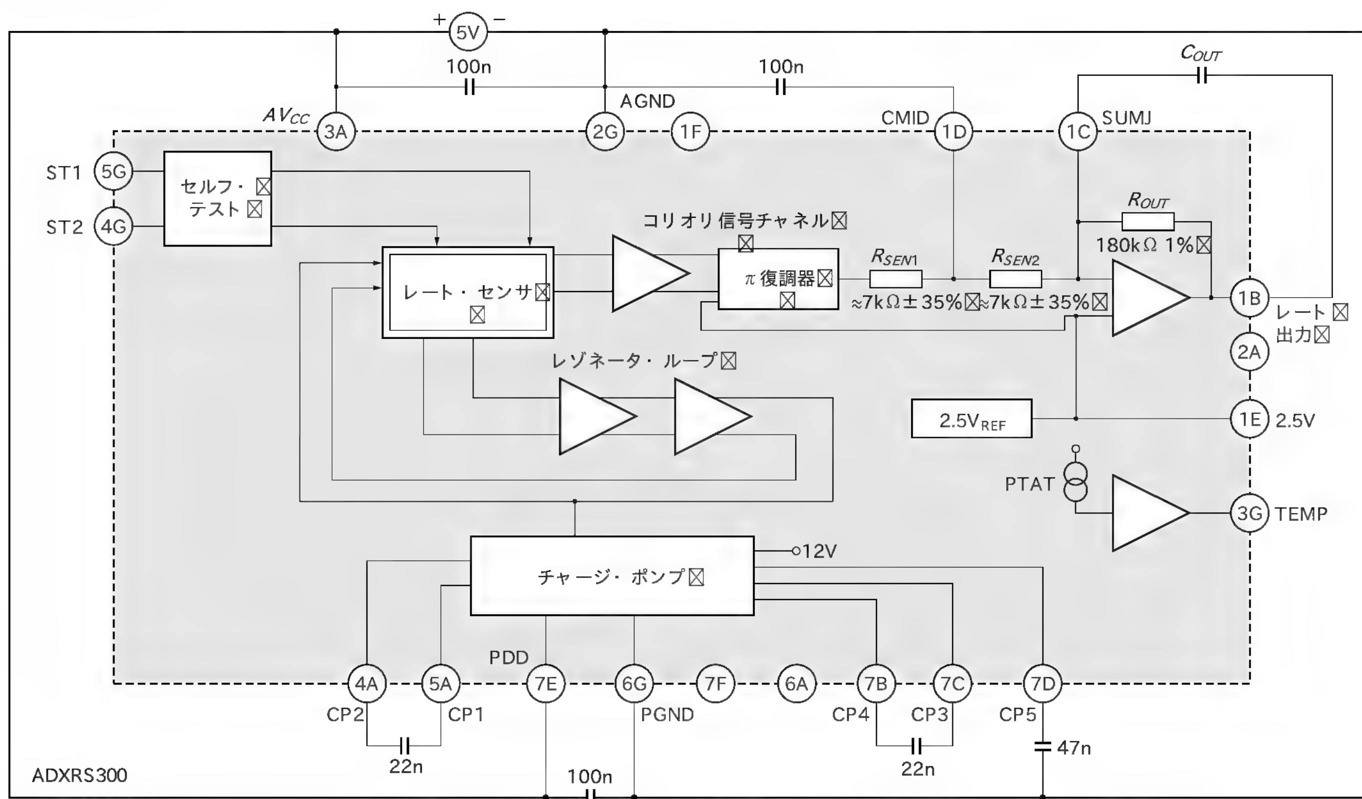


図5[2]　高精度レート・ジャイロADXRS300の機能ブロック図

また，足首角度が床面と並行でないと，床反力がZMPの移動に対応できず，転倒の原因になります．これも足裏感圧センサのデータをもとに足首の角度サーボを正しく制御すれば回避できます．

二足歩行ロボットの制御は，「倒れない歩行アルゴリズム」が基本です．その上でロボットが石を踏んだり，段差に直面したり，ほかのロボットから衝撃を与えられて転倒しそうになったとき，これらのセンサを駆使して臨機応変にバランスを維持します．そのためにどのような制御を行ったらよいのか．これは筆者のチームにとってもこれからの課題です．

引用文献

(1) アナログ・デバイセズ；ADXL311データシート
(2) アナログ・デバイセズ；ADXRS300データシート

よしだ・こうさく

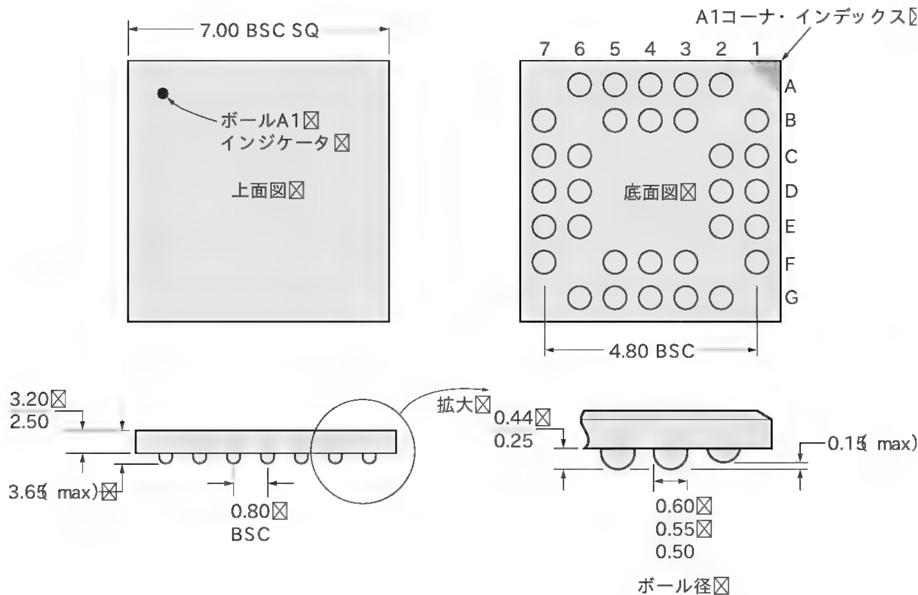


図6[2] レート・ジャイロADXRS300のパッケージ(32ピン・チップ・スケールBGA)

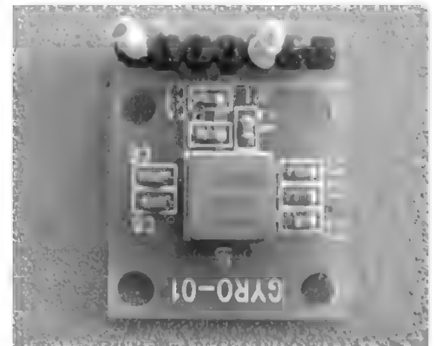


写真2 ジャイロ基板(ADXRS300EB)

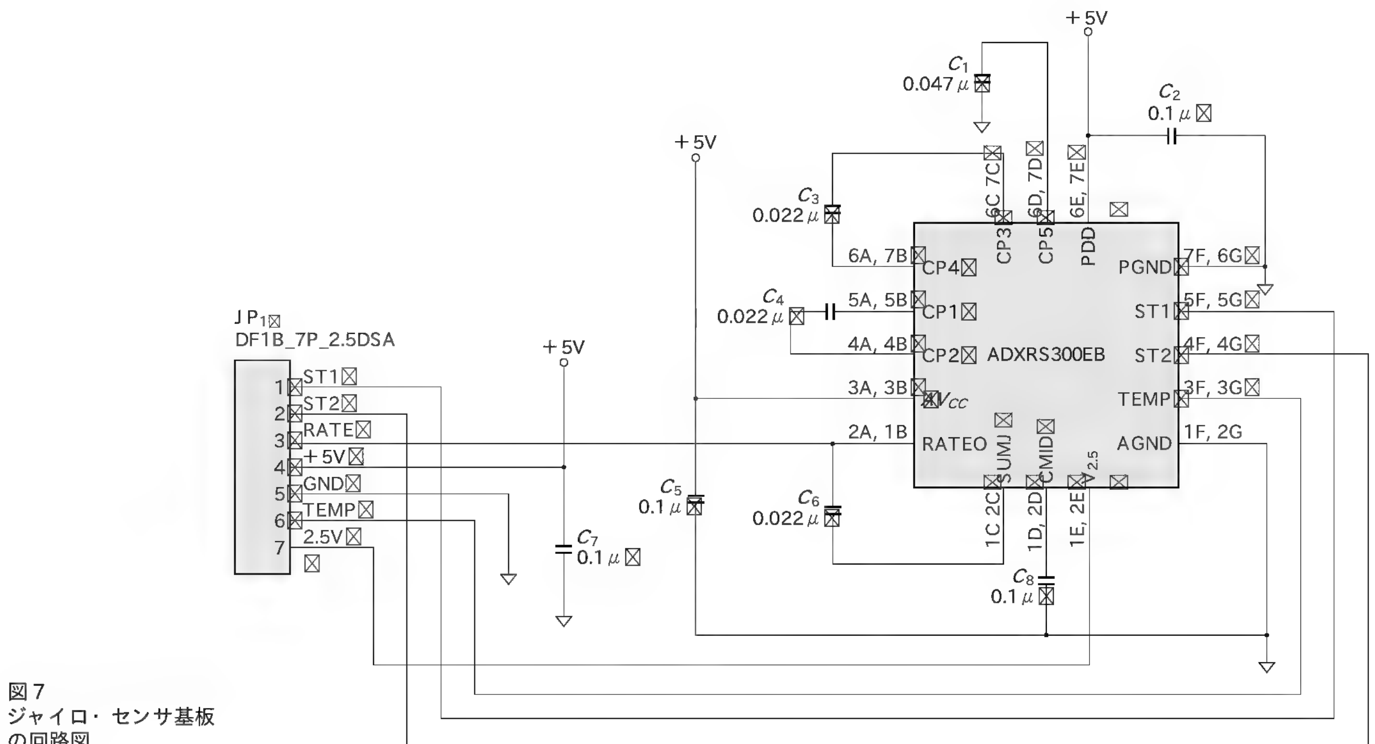


図7 ジャイロ・センサ基板の回路図

Appendix 4

ロボットの電源設計と電池の選択



吉田 幸作

● ロボットに必要な電源

電源設計は，二足歩行ロボットの性能を左右する重要なポイントです．商用電源は使えないので電池を搭載することになります．電池の重量はロボットの重量に加算されるので，できる限り軽くしたい．しかし，1 回の充電で動作可能な時間は長くしたい．したがって，どこで手を打つかが電源設計のポイントになります．

1回の充電で動作可能な時間はなかなか公表されませんが，ASIMOで 30分前後，筆者が試作したWSGH-1で 10分前後です．

どんな種類の電池をどれくらい積むか，これが思案のしどころです．表Aはプロローグで紹介したロボットの電池の種類を集計したものです．

急速放電特性が良くないリチウム電池は 1件にすぎません．ロボットのアクチュエータは高トルク負荷がかかると大電流が流れますが，リチウム電池は大電流特性が良くありません．

● ニカド電池 vs ニッケル水素電池

ロボットの電源として最有力候補は，ニッケル水素電池とニカド電池です．表Aではニッケル水素電池が圧倒的に多く採用されています．これは電池容量/重量比の良さが評価された結果でしょう．市販の単3電池で比較すると，

ニッケル水素電池： 2300mAh / ニカド電池： 1000mAh

と，電池容量で 2倍以上の差があります．加えてニカド電池は環境への配慮という点で逆風下にあります．電機メーカの多くが「200x年までにカドミゼロ」という環境目標を掲げています．

圧倒的にニッケル水素電池に軍配が上がりそうですが，ニカド電池には一つ優れた特性があります．

● 大電流負荷に強いニカド電池

筆者が試作に使ったのは写真Aのニカド組み電池です．単3電池を5本パックにした 5N-700AACL（三洋電機製）です．公称 6V/700mA，電池容量は最新のニッケル水素電池の 1/3以下です．

ニカド電池の唯一のメリット，それは依然としてほかの電池を寄せ付けないすぐれた大電流特性です．図Aはニカド電池の大電流放電特性です．写真Aの組み電池の負荷抵抗を∞，10Ω，9Ω，…，1Ω，0.5Ω，0.33Ω，0.25Ωと変化させたときの端子電圧を測定してグラフ化しました．ニカド電池は無負荷から 20Aまでほぼ 0.25Ωの定抵抗ラインの上に乗っています．

図Bはニッケル水素電池の大電流放電特性ですが，6A程度から急速に端子電圧が低下，10Aでは端子電圧は 2Vまで低下します．

二足歩行ロボット WSGH-1の負荷電流は平均2～3A程度ですが，高トルクを必要とする局面では瞬間的に 6Aを超える大電流が流れます．この電流を供給できるかどうかが，ロボットを安定駆動させるポイントになります．

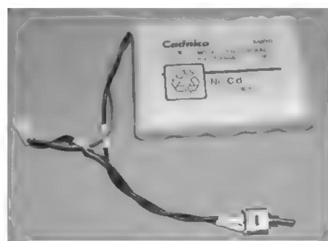
● 安定回路を搭載するときはニッケル水素電池

ロボットのサーボ駆動電源も安定化回路を採用する場合にはニッケル水素電池に軍配が上がります．RCサーボ・モータを使ったロボットではサーボ電流は最低でも 10A程度を見ておく必要がありますが，単3ニッケル水素組み電池でも並列接続すれば 10A程度の負荷電流を供給できます．

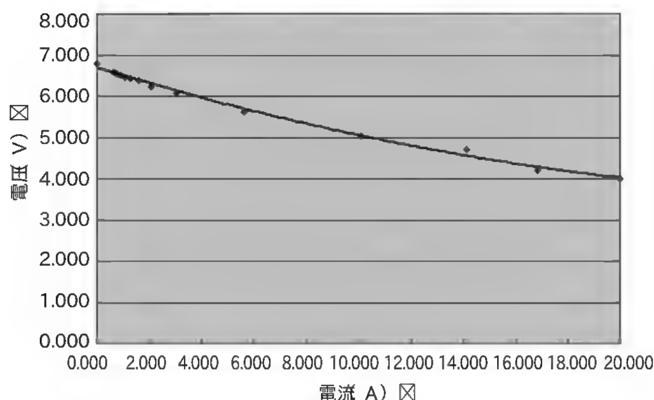
6本直列にした組み電池の出力（公称 7.2V）をステップダウン・レギュレータで 6Vにします．出力電流 10A以上のレギュレータ設計にはひとくふうを要しますが，サーボ系の電源を安定化するメリットは大きいと思います．

表A
ロボットの電池の種類

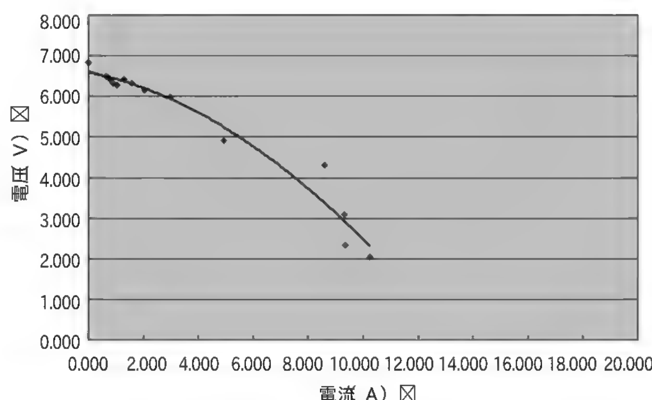
電池の種類	採用ロボット数
ニカド電池	2
ニッケル水素電池	7
リチウム電池	1
鉛蓄電池	1



写真A　ニカド組み電池（三洋電機）



図A　ニカド電池の大電流放電特性



図B　ニッケル水素電池の大電流放電特性

ANSI-Cで記述されたアルゴリズムをFPGAでハードウェアにする

新発想のツール「Impulse C」を使ったソフトウェアのハードウェア化手法

倉重 克己

本稿では，ANSI-Cで記述されたアルゴリズムをハードウェア化でき，同時にハード/ソフト協調設計(コデザイン)も可能にするImpulse C/CoDeveloperを紹介する．

Impulse C/CoDeveloperは，組み込み設計支援を強く意識したツールであり，既存のWindowsベースの開発環境に自然に入り込む方式を取る．CoDeveloper自身もGNU MinGWベースの簡単な開発デバッグ環境を提供している．ハード/ソフトのインターフェース部を隠ぺい化する手段も提供しており，ソフトウェア・エンジニアがハードウェアも含めてシステムを設計する道を大きく開く．

ハードウェア設計を対象にしていても，主たるユーザ層をソフトウェア・エンジニア，または組み込み設計者においている点が，従来のEDAツールと大きく異なる．もちろん，このことはハードウェア・エンジニアにとっても悪いことではない．

C言語での設計とImpulse C/CoDeveloperの概要

まず，このようなツールが誕生する背景に少し触れておく．現在のソフトウェア設計者，ハードウェア設計者は次のような問題を抱えている．

● ソフトウェア設計者のしごと

世の中のほとんどすべての電気製品にはCPUが入り，今やPCやWS上のプログラミングのほうがマイナになってしまうのではと思われるほど，組み込み設計が重要になっている．

それゆえに，必ずしも標準化して抽象化されたものではないむきだしのハードウェアと対峙しなくてはならない場面が多くなり，ソフトウェア設計者もハードウェア設計に深く関わる必要性が出てきている．

また，ソフトウェア処理のボトルネック部をハードウェア化したいという希望も多くあるが，ソフトウェア設計者の設計環境から，それを可能にする容易な手段がない．

● ハードウェア設計者のしごと

RTL設計で組み合わせ回路のゲートの塊の設計から開放されたと思ったのも束の間，ムーアの法則によって回路規模は拡大を続け，それにともない，果てしない高機能化(複雑化)と短納期を求められ，設計者は今やレジスタの塊と日夜格闘している．

昨今，この問題を解決するため，抽象度をもう一段上げたソフトウェア設計と同様のアルゴリズム水準での手法を可能にする手段がSystemCをはじめとして議論され，導入が検討され始めている．

● ソフトウェア設計者，ハードウェア設計者共通のしごと

伝統的には，システム設計者の判断で設計をハード/ソフトに分割し，それぞれの担当が別々に設計して両者がある程度の完成度になったときから統合デバッグを始めていた．

この手法では，ハード/ソフトのインターフェースや統合スループットの問題が後で判明し，その時点からの手戻りは，現在の多くの複雑なシステムでは現実的な手段ではなくなっている．そのため，最初からのソフト/ハードの協調設計が強く求められている．

こういった問題を解決するためのアプローチには，さまざまな方法が考えられ，現実にC言語ベースのツールが出ている．CoDeveloperは，特に「ソフトウェア設計者がハードウェア設計に入り込める手段を提供すること」を第一の目標として企画された．そのため，C言語から仕様を引いたり加えたといった新しい言語ではなく，ソフトウェア設計者の仕事上の母国語ともいうべき標準のANSI-Cをそのまま使い，C言語の知識をハードウェア設計にそのまま再利用できるようにした．

当然，過去のC言語による設計資産も再活用しやすく，一からの設計ではなく，確立されているアルゴリズムをハードウェア化するといった道も開ける．

図1でLagacy C algorithmとして示している部分が過去の資産である．なお，図1で網かけした部分はImpulse Cでの設計をサポートするためのCoDeveloperの要素である．

Impulse Cの仕様

ハードウェアを生成するにあたり，ANSI-Cにハードウェア表現を可能にする並列動作，構造記述などの仕掛けを加えなくてはならない．Impulse Cで採用されている手法はSystemCと同様である．

SystemCが標準ISO C++を使い，クラス・ライブラリを加えて実現しているように，Impulse Cでは標準のANSI-Cに新

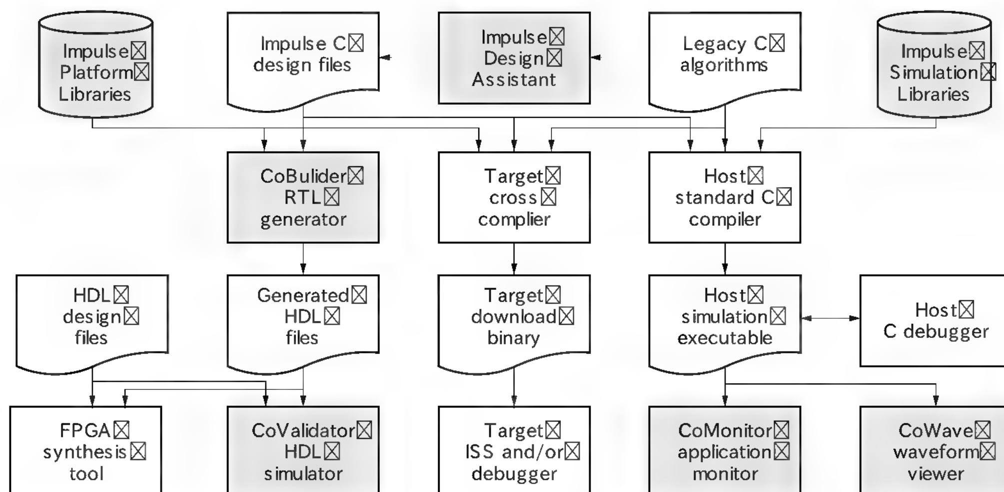


図1
CoDeveloperのツール構成
※網かけの部分はCoDeveloperの要素

表1　Impulse CとSystemCの対比

	Impulse C	SystemC
言語	ANSI-C	ISO C++
ハードウェアサポートのための拡張方法	ANSI-C上で新しい型と組み込み関数定義	C++上で新しいクラス・ライブラリ定義
開発システム	CoDeveloper	各種
ターゲット	おもに組み込み	現在LSI設計が主

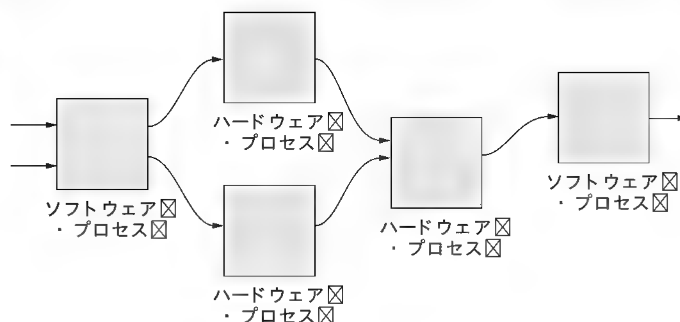


図2　CSPの模式図

しい「型定義」と「組み込み関数定義」のライブラリを加えてサポートしている．その拡張仕様が「Impulse C」と名付けられている．そして，Impulse Cで表現されたハード/ソフト協調設計のシミュレータと動作合成を含む開発環境がCoDeveloperである．表1にImpulse CとSystemCの対比を示す．

また，PSP（Platform Support Package）と呼ばれるターゲットのバス仕様を記述するライブラリが完備されている場合には，CoDeveloperの動作合成ツール（CoBuilder）は，単にハードウェア化する部分の動作合成（VHDL）だけでなく，ハード/ソフトのインターフェースに関わる部分のハードウェア部（VHDL）とソフトウェア部（C）も同時に生成する．PSPにより従来避けて通れなかっためんどうなハード/ソフトのインターフェース部が隠ぺいされ，プロセスという抽象的なソフト/ハード共通概念でシステム全体を扱えるようになり，ソフトウェア・エンジニアも容易にハードウェア設計に参加できるようになる．

Impulse Cの仕様，CoDeveloperの動作合成，そしてPSPの考えは米国ロスアラモス国立研究所で研究開発されたStreams-C環境を元にしている．

Impulse Cでのシステム・モデリング

Impulse Cでは，

　システム＝独立したいくつかのシーケンシャル・プロセスが並列走行しながら，必要に応じて通信して所定の目的を果たすもの

と考える．

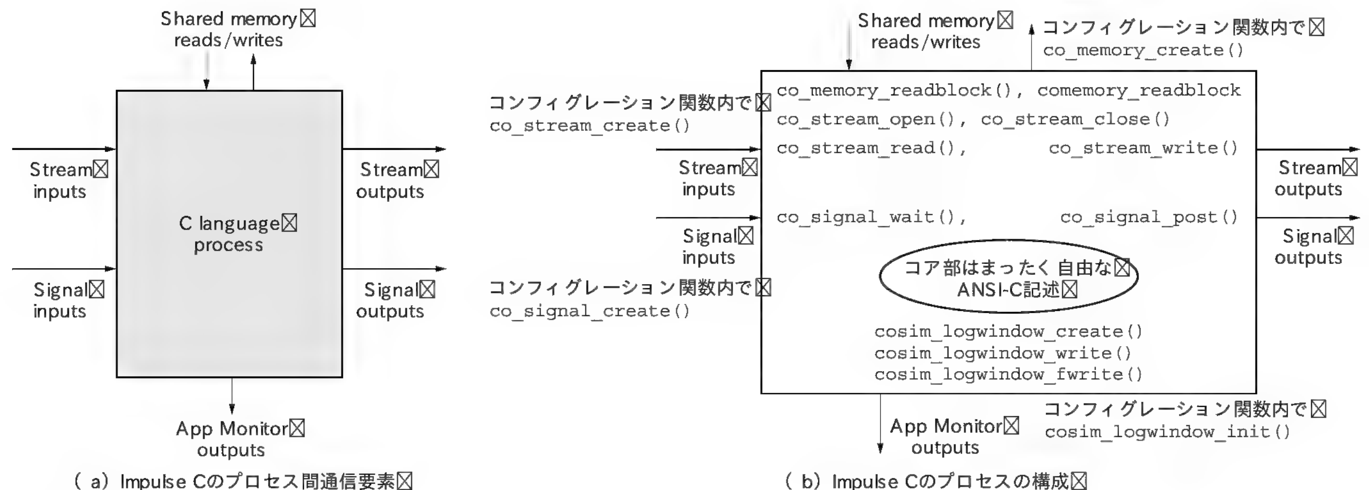
この考えをCSP（Communicating Sequential Process）と呼ぶ．Impulse Cでの各プロセスは「シーケンシャル」なので，C言語で記述可能な通常のサブルーチンと本質的に違いはない．ただ，プロセス間の通信にImpulse Cのライブラリを使う（図2）．

「並列性の粒度」としては，CSPのプロセスは粗粒度に位置付けられるが，CoDeveloperの動作合成モジュールであるCoBuilderは，ループ内部の処理をパイプライン化するより細かな「並列性の粒度」の制御をサポートする．すなわち，Impulse Cの記述は2段階の「並列性の粒度」を表現できる．

1）プロセスとプロセス間通信要素

各プロセスは，図3のようなImpulse Cライブラリで定義されている要素であるストリーム・バッファ〔Stream（FIFO）〕，信号（Signal）と共有メモリ（Shared Memory）を使って互いに通信する．基本的に，モデリング上の制約は，通信にこれらの要素を使うことに限られ，後は自由な記述が許される．

プロセス間の同期はこれらの通信要素で行われる．特に制御付きのストリーム・バッファ〔Stream（FIFO）〕での同期がImpulse Cの特徴である．ストリーム・バッファ長を適切に取ることで淀みなくデータが流れ，自然な同期が取れ，スルー



(a) Impulse Cのプロセス間通信要素

(b) Impulse Cのプロセスの構成

図3 Impulse Cのプロセス

プットを上げることが期待できる手法である．これは，Impulse Cの元になったStreams-Cの名前の由来でもある．

プログラミング・モデルとして，Impulse Cの各プロセスと通信要素は初期化時に定義と同時に生成され，いったん生成されたものが消滅することはない．このようなハードウェアの特性を反映している点は，ソフトウェアにおいて一般のプロセスやスレッドの生成/消滅が自由であることと大いに異なるが，動作シミュレーション時のImpulse Cプロセスは当然スレッドでモデル化される．なお，AppMonitor(CoMonitor)はデバッグのために，プロセス内の変数値をシミュレータに渡し，観測可能にするしかけである．

図3(b)は，実際に使われるImpulse Cの関数を，図3(a)との対比で示したものである．一般的なI/Oアクセスやインターフェース・ライブラリと似た雰囲気があり，ソフトウェア・エンジニアは関数名から，ある程度のことを想像できるかもしれない．

このような，入出力部がImpulse C関数を経由するサブルーチンを，Impulse Cのプロセスとして定義し，これらを相互につなぐことでシステムを表現する．Impulse Cでのハードウェア化はこのプロセス単位で行われる．

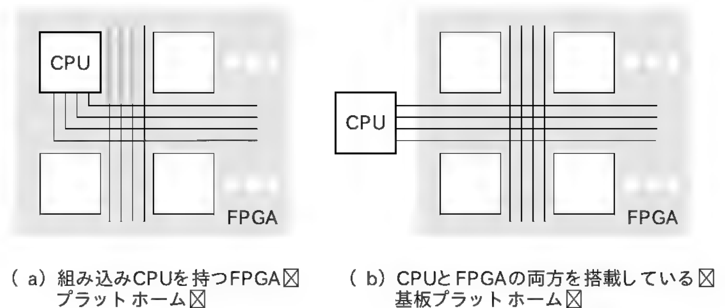
2) コンフィグレーション関数

プロセス間の通信，結合を表現する記述をシステムの構成を表す特別な関数，コンフィグレーション関数で定義する．

コンフィグレーション関数は，システムの最上位でユーザが設計したプロセスと，Impulse Cライブラリで定義してある通信要素をシステム構成に合わせてインスタンス化する．

これは，いってみれば，ハードウェア設計の最上位のネット表現と同じ概念だが，大きな違いは，相互につながれるプロセスはハードウェアとは限らず，ソフトウェアでもありうることであり，この点が重要である．

各プロセスをソフトウェア化するのか，ハードウェア化するのかも，この中で指定されるが，機能検証段階で機能シミュレーションではこの違いは区別されない．ハードウェア化する動作合成時に初めて解釈される．



(a) 組み込みCPUを持つFPGAプラットホーム

(b) CPUとFPGAの両方を搭載している基板プラットホーム

図4 考えられるプラットホーム

3) イニシャライズ関数とプラットホーム指定

イニシャライズ関数は，プログラミング・モデルとしては，コンフィグレーション関数を呼び出し，Impulse Cの各プロセスと通信要素を定義と同時に生成して，実行可能にする．

実装的には，コンフィグレーション関数で構成したシステムを実装へ渡す最終指定をする役割をになう．

すなわち，コンフィグレーション関数を指定して選択するプラットホームへのマッピングを動作合成過程に指示する．Impulse C/CoDeveloperは，設計の実装対象を「プラットホーム」と考え，実装対象のプラットホームの種類を指定する．

「プラットホーム」は，ソフト/ハード両方のプログラミング資源を持つシステムと規定され，現実には「組み込みCPUを搭載可能なFPGA[図4(a)]」と，「CPUとFPGAが実装されており，ハード/ソフト両方のプログラミング・リソースが開放されている基板[図4(b)]」の2種類が対象になる．

CoDeveloperのGUI環境からのコントロールでは，CoDeveloperのダイアログ経由でのプラットホーム指定が優先される．また，合成HDLだけを取り出すGenericモードもある．

現在，CoDeveloperは，組み込みCPUを持つFPGAプラットホームとして，Nios（Altera社）とMicroBlaze（Xilinx社）をサポートしている．2番目の種類の基板プラットホームは，基板上のバス仕様などを統一した規格でライブラリ化する手順をImpulse社が用意する．これにより基板上のプロセスも抽象化されCoDeveloper上で一貫してサポートが可能となる．

4）Impulse Cのプログラム構成

Impulse CはANSI-Cの枠内の仕様なので，プログラムは当然`main`関数を持ち，その下に前述した必須のイニシャライズ関数，コンフィグレーション関数，プロセス関数がくる（表2）．

5）シミュレーションとデバッグ

Impulse CはANSI-Cの一つのアプリケーションなので，ANSI-C準拠のコンパイラやデバッガなど，各種のツールが使用できる．動作合成前の動作検証段階では，設計全体をWindows上のプログラムとしてシミュレーションできる．したがって，ソフトウェア設計や組み込み設計の現場で慣れ親しんでいるWindows上のVisualStudioやCodeWarrior，GNUのIDE環境を，そのまま活用できる（図5）．

プロセスの並列動作を観察する動作シミュレータ（CoMonitor）と，これらの既存環境を組み合わせてImpulse C設計のデバッグに利用できるようプロジェクト・マネージャ（CoManager）はできている．

表2　Impulse Cプログラムを構成する関数

●main関数 ＊システムを起動 xxxxarch＝Initialize関数()； co_execute(xxxxarch)； ＊ほかの部分はテスト・ベンチの一部
●Configuration関数
●Initialize関数
●プロセス関数 ＊ソフト・プロセス関数には自由なテスト記述が可能 ＊プロセス内での内部状態をCoMonitorに渡して観察
●ほかのプロセスとはならない関数 ＊テスト・ベンチの一部 ＊ソフト・プロセスのサブルーチン

実装の過程

1）動作合成とハード化

コンフィグレーション関数でハード化指定をしたプロセス関数は，CoDeveloperの動作合成ツールであるCoBuilderによってRTLレベルのVHDLに動作合成される．CoBuilderに通す前に，ハードウェア化するプロセスに対しては，必要に応じて多少の書き直しが必要になる．定義したImpulse Cでの入出力はそのままに，ハードウェア化できないC言語の表現を，等価な形でハードウェア化できる表現に書き換えなくてはならない（図6，表3）．

CoBuilderの動作合成には，スタンフォード大学，ロスアラモス国立研究所，Impulse社それぞれで開発された三つの強力なオプティマイザが備わっており，品質の良い合成を行う．だが，動作合成の特性を知っておくことで，より少ないリソースで高速に動作させるための表現を追求をすることができる．

Cの`#pragma`で動作合成ユニットCoBuilderにループ内のパイプライン化を指示でき，また分割されるパイプライン・ス

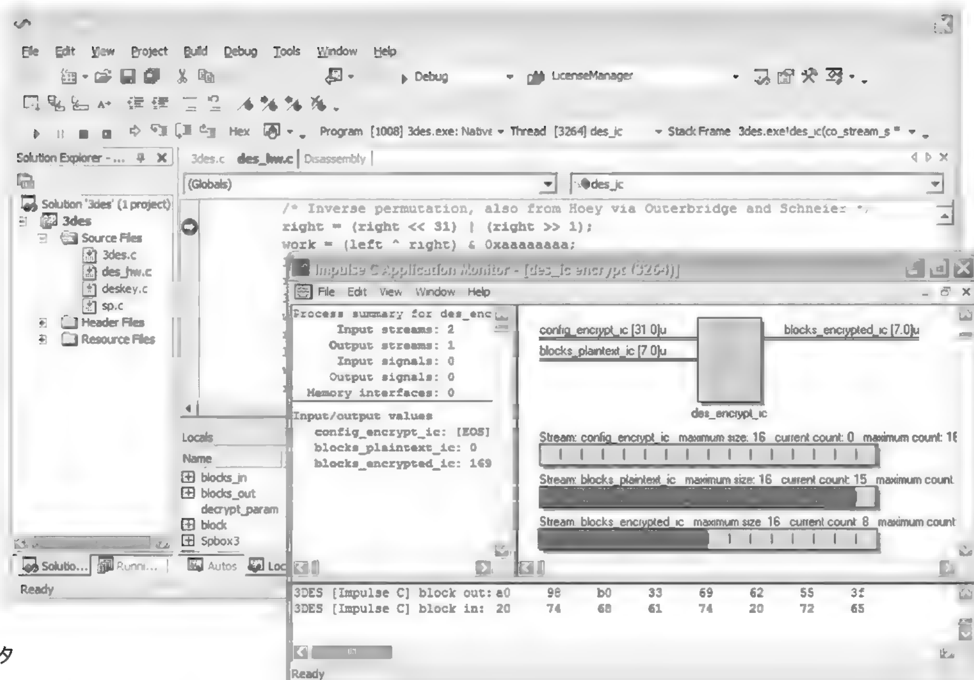


図5
VisualStudio.NET上で動作シミュレータCoMonitorを使ってデバッグ

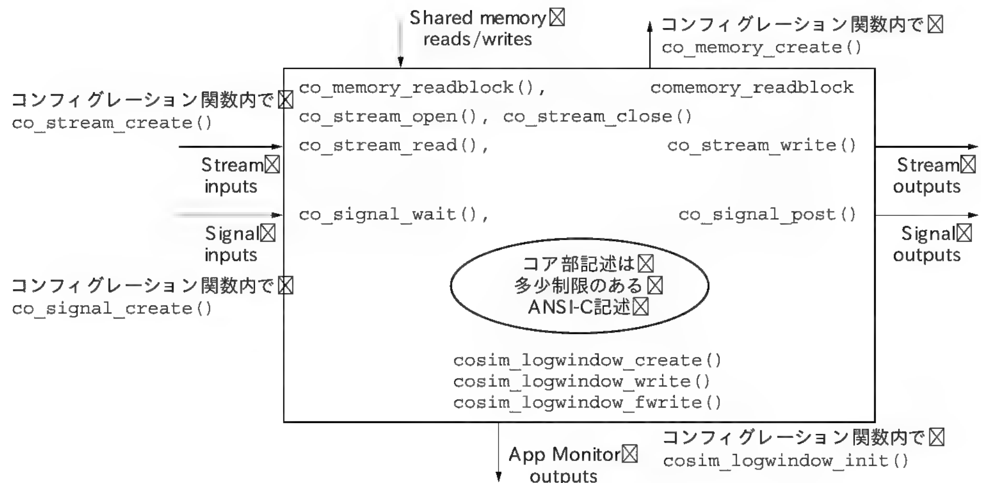


図6
ハードウェア化プロセスのコア部の表現を等価に書き換える

表3　ハードウェア化のための言語上の制約

●再帰的な表現の禁止～ハードウェアは自分自身を呼べない ●プロセス内での，Impulse C関数以外の関数呼び出しの禁止 ～ハードウェアにはスタック・ポインタがない インライン関数，マクロはもちろん使用可能 ●コンパイル時に確定値を持つポインタ以外のポインタ使用禁止 ●配列以外の複合データ型の禁止

テージの最大遅延の指定も可能で，動作速度の制御ができる．

2）ハード/ソフトのインターフェースとPSP

CSPのプロセスは抽象的なものでハード/ソフトの区別はなく，Impulse Cでもそれを踏襲している．それゆえに，高い次元での設計を可能にし，ソフトウェア・エンジニアにハードウェア設計への道を開くが，実装時には，それまで隠れていたハード/ソフトのインターフェースの問題が顕在化してくる．

たとえば，FIFOでプロセスがつながれている場合を想定しよう．プロセス間が，ソフト-ソフトであればFIFOはメモリ上のデータ構造の問題として解決する．また，ハード-ハード間であれば物理的にFIFOバッファを互いに直結すればよい．

しかし，ハード-ソフト間となると，CPUとハードウェアはCPUバスを経由しての複雑なプロトコルでFIFOをエミュレートする必要がある．このような問題を解決し，プロセスをハードウェアとソフトウェアの区別なく同等に扱えるようにするためのインターフェース・パッケージがPSP（Platform Support Package）である．

PSPによりプロセスは，現実は図7中の下図だが，ユーザはインターフェースのない（見えない）図7中の上図のような扱いが可能となる．

PSPが用意されていると，CoBuilderはハード化プロセスのアルゴリズムをVHDLに合成するだけでなく，ユーザの設計に合わせたハードウェア側，ソフトウェア側両方のインターフェースを自動生成してくれる（図8）．

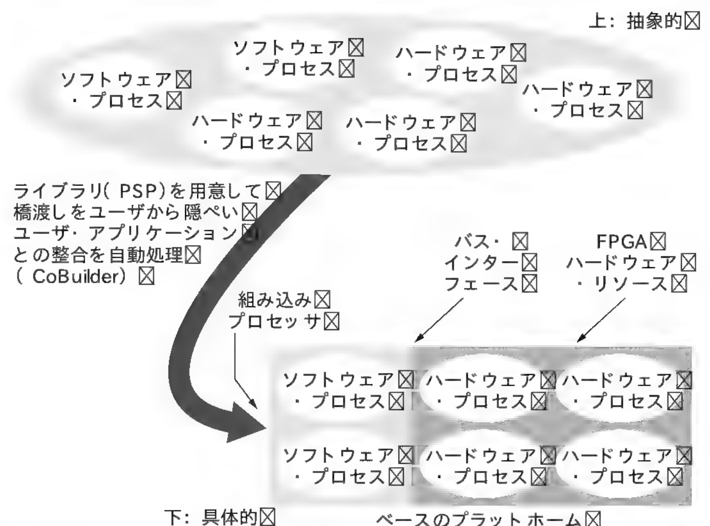


図7　PSPの役割

3）FPGAベンダ・ツールへの設計転送とプロジェクト化

CoBuilderで生成されたRTL-VHDL，インターフェース・モジュールと分離されたソフトウェア部のCコードは，Altera社のQuartusやXilinx社のISEといったFPGAベンダ・ツールのプロジェクトとして登録し，実装の準備を行わなくてはならない．

CoDeveloperは，この過程を半自動化してユーザの負担を軽減している．具体的には，プロジェクト・フォルダの指定をするだけで，QuartusやISEの要求する形で生成したファイル群が指定したプロジェクト・フォルダに転送される（図9）．

これにより，ユーザはインターフェースに関して必要となる知識とわずらわしさから解放される．

あとは，プロジェクトに関わる設定，NiosやMicroBlazeのメモリやペリフェラルに関わる設定，ピン配置など，お決まりの設定を行えば完成である．NiosやMicroBlazeが実装できるFPGAを搭載したボードがあれば，CoBuilderの合成出力から約

20分でFPGAコンパイル前まで作業を進めることができる．

4）VHDLモジュール生成ツールとしての活用

CoDeveloperは「プラットホーム」に対するハード/ソフト協調設計（コデザイン）への適用だけでなく，C言語で記述されたアルゴリズムをVHDLのRTLに変換する機能もサポートしている．

プロセス間通信要素の仕様は公開されており，Streamバッファ（FIFO）は対象のプロセスから変換されたVHDLと明確に分けて存在し，生成したモジュールを取り出すのは困難なことではない．ポート仕様がわかっているので，ほかのモジュールとの結合やシミュレーションは容易である．

FPGA設計の場合は，たとえ組み込みCPUのNiosやMicroBlazeを最終的には使わなくても，いったんいっしょにFPGAに組み込むことで，これらのCPUは「組み込まれた」FPGA設計デバッグ環境として機能する．

そこで自由にC言語でテスト・コードを記述し，FPGAをデバッグすることができる．FPGAのハードウェア部が完成した後にCPUを外せば済むことで，デバッグ効率が飛躍的に上が

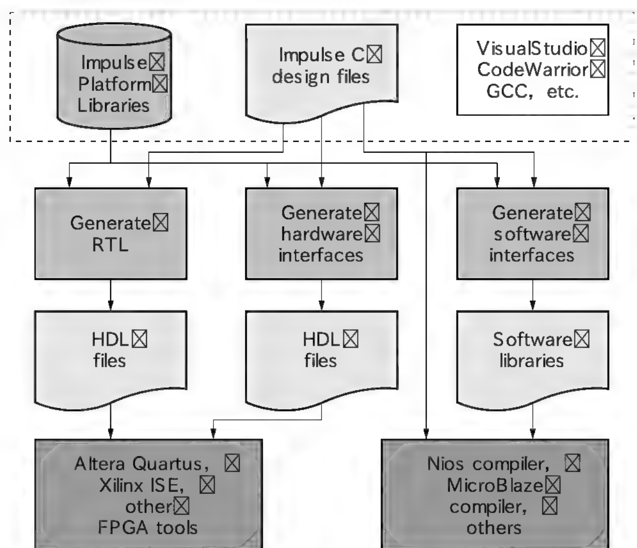


図8　CoBuilder動作合成エンジンのインターフェース生成

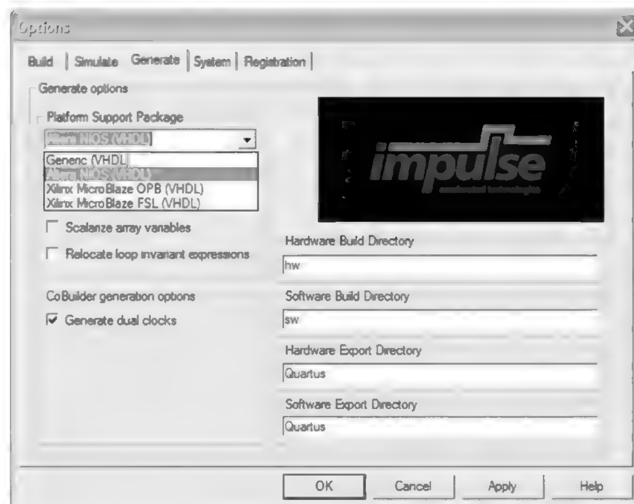


図9　プロジェクトの転送

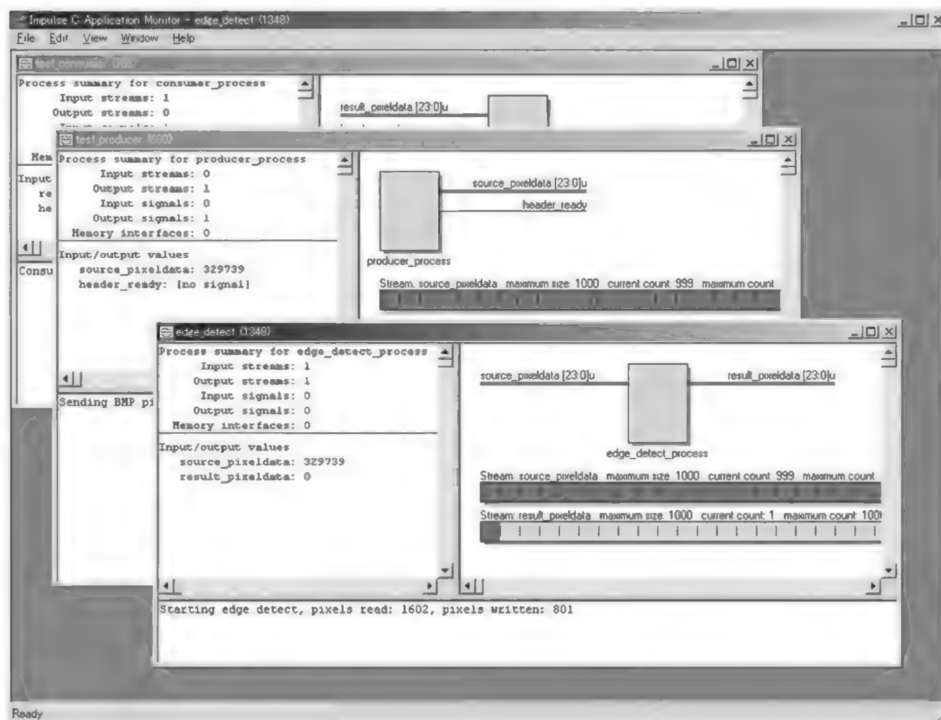


図10　EdgeDetectシミュレーション

ることが期待できる．

次に示す二つの実例は，このような手法によるものである．

Impulse C/CoDeveloper 適用例

代表的な応用と思われる，画像処理と暗号システムの例を示す．図4（a）のタイプのプラットホームで，すでにPSPが製品化されている組み込みCPUであるNiosとMicroBlazeを使用した例を1例ずつ紹介する．

● 画像処理への適用例――エッジ検出

Impulse C/CoDeveloperを使用しての，仕様の検討からFPGAへの実装の流れを，実際に起こりそうなストーリ仕立てで実例で説明する．この例はAltera Stratix 1S10で確認された．

1）目的に合ったフィルタ方式とアルゴリズムの選択

エッジ検出画像フィルタとして多くの種類があるが，目的に合ったものはどれか選択しなくてはならない．同時に実現のアルゴリズムを検討しなくてはならない．Impulse Cで記述すると，この段階から実装までが終始一貫して少数のツール上で，慣れ親しんだANSI-Cで実現可能である．Impulse Cの記述は大枠のCSP的な処理フローを決め，コンフィグレーション関数定義と必要なプロセス関数群の一例を作ると，後はそれがテンプレートとなり，アルゴリズム部は図3（b）のコアに対応するので，まったく自由にフィルタの種類とその処理アルゴリズムの検討ができる．

1次微分（グラディエント）フィルタ・システムとそのアルゴリズムがデバッグされ，次にある2次微分（ラプラシアン）によるものを試し比較した．2次微分版は1次微分版で，アルゴリズム・コア以外の部分がテンプレート化されていたので，5分で完了した（図10）．

リスト1～リスト3に，これまでのImpulse Cソースの一部を示し詳細に解説する．以下の説明と併せて読んでもらいたい．

ここで少し，今回試した画像フィルタに触れておく．

画像フィルタの基本は注目している画素（ピクセル）と隣接する8画素を加えた3×3に切り出した9画素の線形演算を基本

リスト1　**main**関数，コンフィグレーション関数，イニシャライズ関数

```
int main(int argc, char *argv[])
{
    co_architecture my_arch;
    void *param = NULL;
    int c;

    printf("Impulse C is Copyright 2003 Impulse Accelerated Technologies, Inc.¥n");

    my_arch = co_initialize(param);
    co_execute(my_arch);

    printf("¥n¥nApplication complete. Press the Enter key to continue.¥n");
    c = getc(stdin);
    return(0);
}

co_architecture co_initialize(int param)
{
    return(co_architecture_create("edge_detect_arch","Generic_VHDL",config_edge_detect,(void *)param));
}

void config_edge_detect(void *arg)
{
    co_stream source_pixeldata, result_pixeldata;
    co_signal header_ready;
    co_process producer_process;
    co_process edge_detect_process;
    co_process consumer_process;

    IF_SIM(cosim_logwindow_init();)

    source_pixeldata = co_stream_create("source_pixeldata", UINT_TYPE(24), BUFSIZE);
    result_pixeldata = co_stream_create("result_pixeldata", UINT_TYPE(24), BUFSIZE);
    header_ready = co_signal_create("header_ready");

    producer_process = co_process_create("producer_process", (co_function)test_producer, 2,
                                         source_pixeldata, header_ready);
    consumer_process = co_process_create("consumer_process",(co_function)test_consumer, 2,
                                         result_pixeldata, header_ready);
    edge_detect_process = co_process_create("edge_detect_process", (co_function)edge_detect, 2,
                                         source_pixeldata, result_pixeldata);

    co_process_config(edge_detect_process, co_loc, "PE0");    // Assign processes to hardware elements
}
```

リスト 2　プロセス関数 EdgeDetect 1次微分版

```
void edge_detect(co_stream pixels_in, co_stream pixels_out)
{
    IF_SIM(int pixelsread = 0;)
    IF_SIM(int pixelswritten = 0;)
    int currentpos;
    int idx = 0;
    int addpos;
    short pixeldiff1, pixeldiff2, pixeldiff3, pixeldiff4;
    short pixelC, pixelN, pixelS, pixelE, pixelW;
    short pixelNE, pixelNW, pixelSE, pixelSW;
    short pixelMag;
    co_uint8 bytebuffer[BYTEBUFFERSIZE][3];
    co_uint8 nByte;
    co_uint8 nByteMag[3];
    co_uint24 nPixel;
    co_uint2 clr = 0;

    IF_SIM( cosim_logwindow log; )
    IF_SIM ( log = cosim_logwindow_create("edge_detect"); )

    co_stream_open(pixels_in, O_RDONLY, UINT_TYPE(24));
    co_stream_open(pixels_out, O_WRONLY, UINT_TYPE(24));
        ........

    IF_SIM(cosim_logwindow_fwrite(log,
          "Starting edge detect, pixels read: %d, pixels written: %d¥n", pixelsread, pixelswritten);)

    while ( co_stream_read(pixels_in, &nPixel, sizeof(co_uint24)) == co_err_none ) {
        IF_SIM(pixelsread++;)
        bytebuffer[addpos][REDINDEX] = nPixel & REDMASK;
        bytebuffer[addpos][GREENINDEX] = (nPixel & GREENMASK) >> 8;
        bytebuffer[addpos][BLUEINDEX] = (nPixel & BLUEMASK) >> 16;
        addpos++;
        if (addpos == BYTEBUFFERSIZE) addpos = 0;
        currentpos++;
        if (currentpos == BYTEBUFFERSIZE) currentpos = 0;

        for (clr = 0; clr < 3; clr++) {  // Red, Green and Blue
            pixelC  = bytebuffer[B_OFFSETADD(currentpos,0)][clr];
            pixelN  = bytebuffer[B_OFFSETADD(currentpos,WIDTH)][clr];
            pixelS  = bytebuffer[B_OFFSETSUB(currentpos,WIDTH)][clr];
            pixelE  = bytebuffer[B_OFFSETADD(currentpos,1)][clr];
            pixelW  = bytebuffer[B_OFFSETSUB(currentpos,1)][clr];
            pixelNE = bytebuffer[B_OFFSETADD(currentpos,WIDTH+1)][clr];
            pixelNW = bytebuffer[B_OFFSETADD(currentpos,WIDTH-1)][clr];
            pixelSE = bytebuffer[B_OFFSETSUB(currentpos,WIDTH-1)][clr];
            pixelSW = bytebuffer[B_OFFSETSUB(currentpos,WIDTH+1)][clr];

            pixeldiff1 = ABS(pixelSE - pixelNW);
            pixeldiff2 = ABS(pixelNE - pixelSW);
            pixeldiff3 = ABS(pixelS - pixelN);
            pixeldiff4 = ABS(pixelE - pixelW);
            pixelMag = MAX4(pixeldiff1,pixeldiff2,pixeldiff3,pixeldiff4);

            nByteMag[clr] = (co_uint8) pixelMag;
        }
        nPixel = nByteMag[REDINDEX] | (nByteMag[GREENINDEX] << 8) |
                    (nByteMag[BLUEINDEX] << 16);

        co_stream_write(pixels_out, &nPixel, sizeof(co_uint24));
        IF_SIM(pixelswritten++;)
    }

    IF_SIM(cosim_logwindow_fwrite(log,
        "Completed edge detect, pixels read: %d, pixels written: %d¥n", pixelsread, pixelswritten);)
........
    co_stream_close(pixels_in);
    co_stream_close(pixels_out);
}
```

プロセス関数edge_detect

画像データ格納リング・バッファ

CoMonitorにこのプロセスのモニタリング・ウインドウを開く

streamの方向を指定して読み書きできるように初期化

ストリームから原画1画素分のデータを読み込み(RGBいっしょの24ビット)

3×3画素データ(色毎)
pixelCが注目画素
pixelCからの方位で他の画素の名前付け

画像フィルタリング・アルゴリズムの核
この部分が処理スピードを決定する
必要があれば実装時にプラグマ指定でループ内をパイプライン化

ストリームへ処理済み1画素分のデータを出力

CoMonitor上のこのプロセスのモニタリング・ウインドウに処理されたストリーム・データの読み込み，書き込みカウントを表示

streamのクローズ処理

とする．縦横 $m\times n$ の画像とし左下を原点として0番を振り，右へ番号を増やしながら $n-1$ に到達すると，0番の上に一つ進んで n 番として，これを繰り返す．最後の画素番号は $m\times n-1$ で右上となり，3×3の画素位置番号は注目画素を k とすると，図11のようになる．

A）1次微分(素朴なグラディエント)フィルタ

注目画素を挟む画素の縦，横，斜め2方向の画素ペアについて，ペア間の計四つの差分を求めそれぞれ絶対値を取る．四つ

リスト 3　2次微分(変形8近傍ラプラシアン)のアルゴリズムの核

```
pixelMag = pixelC << 3;
pixelMag -=  pixelSE;
pixelMag -=  pixelNW;
pixelMag -=  pixelNE;
pixelMag -=  pixelSW;
pixelMag -=  pixelS;
pixelMag -=  pixelN;
pixelMag -=  pixelE;
pixelMag -=  pixelW;
if (pixelMag < 0)
        pixelMag = 0;
```

この画像フィルタリング・アルゴリズム☒の核を1次微分版と入れ替える☒実装時にはプラグマ指定でループ内をパ☒イプライン化した☒

$k-1$ $+n$	k $+n$	$k+1$ $+n$
$k-1$	k	$k+1$
$k-1$ $-n$	k $-n$	$k+1$ $-n$

図11
3×3画素フィルタリング・ウインドウ


(a) 原画


(b) 1次微分フィルタ結果

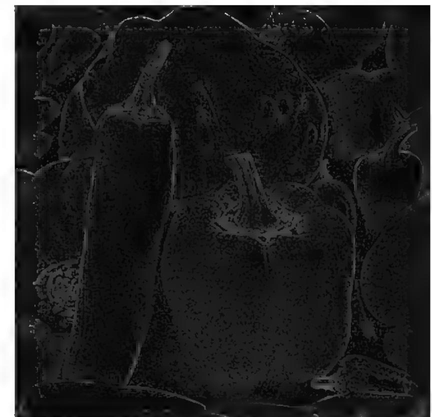

(c) 2次微分フィルタ結果

図12　画像フィルタリングの例

のうちの最大を注目画素に対する処理結果とする．これをRGBそれぞれに対して行う．

B) 2次微分(変形8近傍ラプラシアン)フィルタ

一般的な8近傍ラプラシアン・フィルタは，注目画素の値を8倍し，それからその画素を取り巻く8画素の値の合計を引き，それを注目画素に対する処理結果とする．ある目的に周りより明るい点だけを抽出する，結果がプラスになったものだけ取り，ほかは0とする．これをRGBそれぞれに対して行う．

図12(b)，図12(c)の結果から2次微分(ラプラシアン)のほうが目的に適切であると判断し，これを選択することにした．

2) 並列化の検討

次に，目的に合った効率的なプロセスの並列化を検討する．フィルタ方式とアルゴリズムの選択には図13のモデルを使った．

すなわち，一つのハードウェア・プロセス edge_detect_process で，24ビット(8ビット×3)の3原色データのストリームを受信しながらフィルタリングを行った結果の24ビット3原色データを送り出す処理を行うモデルである．

edge_detect 関数内部を並列化はできない．Impulse Cの取るCSPモデルのプロセス単位はシーケンシャルであり，子プロセスはない．並列化の導入は，edge_detect 関数の行っている処理を分割して複数のプロセス関数に分けることである．

RGB3原色の各色単位の並列化など，いろいろと考えてシミュレーションを行ったが，検討の結果，edge_detect 関数の行っていた処理を二つのプロセスに分ける次の処理がハードウェア化された．

「データ入力，3×3のピクセル単位の切り出しに必要なバッファリングと必要なカラム・データの取り出し」を行う prep_run 関数と，「フィルタリング処理と結果出力」を行う filter_run 関数が用意され，edge_detect 関数と置き換えられた．

その結果，図14のようなパイプライン処理が実行され，処理スピードの向上が期待できる．

3) 実装

●書き換え

シミュレーション完了を持って，実装に移すのにあたり，若干ソフトウェア側のコードの書き直しが必要になった．ターゲットはStartixが搭載されたAltea社のNios開発キット(1S10)ボードである．このボードの持つCompact Flashディスク機能を使えば手段はあったはずだが，不勉強でWindows上と同じような大きなビットマップ・ファイル形式のテスト・データ，変換データが取り扱えず，検証とデータに別の手段を取る必要があった．

32×32の画像サイズをNiosプログラムの一部として，これを敷き詰めて512×512とした．検証は変換されたデータの一部32×32を文字変換してRS-232-C経由でコンソール上で確認した．

Windows環境で十分な検証が済んでいるからこの程度の検証で安心である．

一般的に，実装に近くなればなるほど，デバッグのためのリ

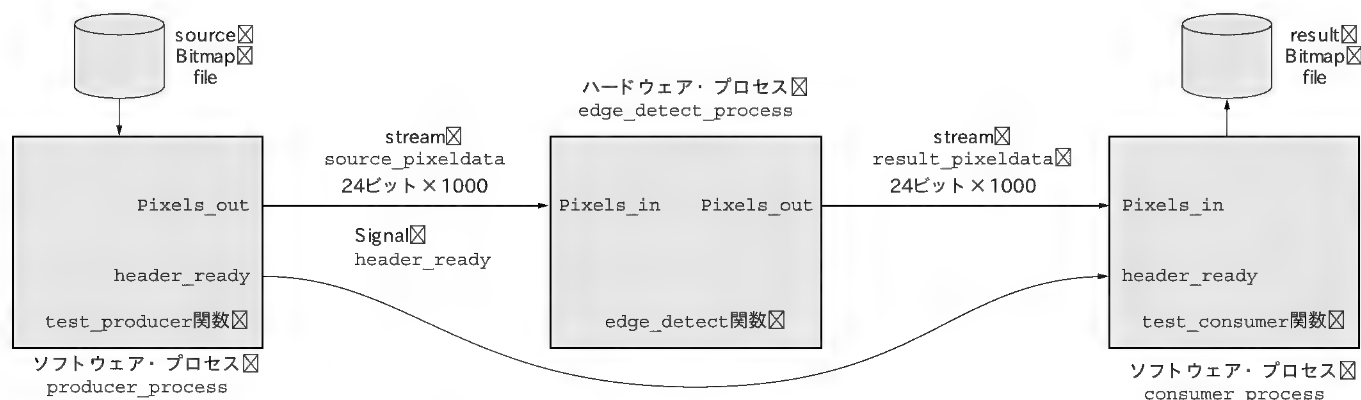


図13　画像フィルタ・シミュレーション・モデル

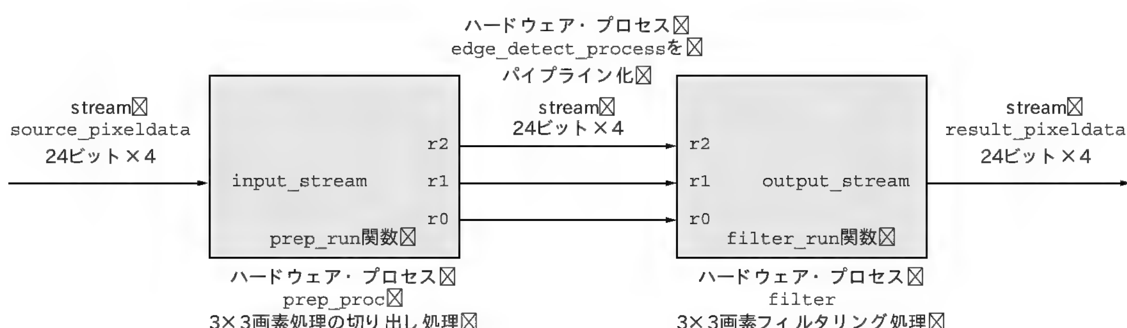


図14
プロセスを分割し，
並列化を検討

```
------------------------------------------------------------
;Fitter   Summary                              ;
------------------------------------------------------------
; Fitter Status ; Successful - Tue Mar 02 12:02:26 2004 ;
; Compiler Setting Name    ; timg              ;
; Top-level Entity Name    ; timg               ;
; Family                   ; Stratix              ;
; Device                   ; EP1S10F780C6     ;
; Total logic elements     ; 4,972 / 10,570 (  47 % )  ;
; Total pins               ; 108 / 426 (  25 % )      ;
; Total memory bits        ; 59,780 / 920,448 (  6 % ) ;
; DSP block 9-bit elements ; 0 / 48 (  0 % )           ;
; Total PLLs               ; 0 / 6 (  0 % )           ;
; Total DLLs               ; 0 / 2 (  0 % )           ;
------------------------------------------------------------
```

図15　必要としたFPGAのリソース（Niosを含む）

ソースや手段が少なくなるうえに情報が不足しがちでめんどうが増す．デバッグの繰り返し時間も飛躍的に長くなる．抽象度の高い所で，最大限できることをキッチリやる姿勢が大切である．

●動作合成とビット・ファイル生成

CoBuilderは120行程度のハード化C言語ソースを1500行程度のVHDLに1分程度で合成し，NiosのPSPによりインターフェースも生成した．それと同時にNios上で走行するC言語部分も適切に切り分けられた．指定ディレクトリ（図9）にQuartusのプロジェクトに合った形で合成されたファイル群も転送された．SOPC BuilderでNios下のバス（Avalon）に結合してシステム構成を完了し，ピン配置などの設定後，30分程度のコンパイル時間でビット・ファイルを得た．Quartusのレポートの一部を図15に示す．Niosとペリフェラルを入れて約50%のリソースの使用率である．

動作速度としては，2クロック・サイクルで1ピクセルの処理が実現されている．クロック50MHzでRGB 3×8ビット・フルカラー画素数512×512のフィルタリング処理に約10msを要するが，十分に実用になる水準である．

図16は，Niosが合成されたFPGA上のフィルタから受け取った，32×32画素の処理結果を元に文字変換（明るさが増す方向により大きく複雑な文字を割当）し，コンソール出力したものである．図の縁だけが現れ，正しく処理されている．

ソフトウェア・プロセスとのstream通信によるCPUデータ転送であったsource_pixeldataとresult_pixeldataを，Shared Memory通信に変え，DMA転送版も実装して試した．DMA転送版はCPUデータ転送版よりデータ転送速度が約70倍速く，Impulse CでもDMAの効果が確認された．

●既存のC言語資産の活用――3DES暗号の例

既存のANSI-Cソースを再利用してハードウェア化する例として，現在もっとも使われている暗号システムの一つである3DES暗号アルゴリズムのハードウェア化の例を取り上げる．

この例は，Xilinx Virtex IIデバイス上で動作を確認した．

データ通信で使われる暗号は高速で流れて来る平文ストリーム入力を，暗号化して同率のレートで送出しなくてはならないので高速処理が必要になる．これは逆も同様である．Impulse Cのストリームは，この処理の入出力に適したものと思われる．

1）採用した3DESアルゴリズム

Qualcomm社のエンジニアであるPhil Karn氏が公開し，パブリック・ドメインになっている3DESのソース・コードを使用する（http://www.ka9q.net/）．これは，並列動作向けに開発されたものでなく，通常のCPUでの実行を想定したものである．

2）Impulse Cのモデル

ハードウェア化するため，Impulse Cのプロセス関数に適合させる変更をする必要がある．データ・ブロックとキーの送受信にストリームの読み込み/書き込みを使用したストリーム・ベースの通信にするための最低限の変更を加え，核のアルゴリズムはまったく変更を加えなかった．

CoDeveloperの統合マネージャであるCoManegerには，Impulse Cの通信要素をつなぐポート記述を補助する機能――DesignAssistant機能（図17）があり，旧資産をImpulse C記述へ変更する助けとなる．

「元のCコードによるCPU走行」と，Impulse Cによりハードウェア化した「専用エンジン」との同一データによる比較をするため，この二つのバージョンが一つのシミュレーション対象になるようなシステムをImpulse Cで記述（図18）した．

前例の画像フィルタと同様に，大きなテキスト・ファイルを使って大量のデータを扱うシミュレーション（図19）で暗号変換の動作検証をした．

3）実装

実装は，同一FPGA上で，ハードウェア化した3DES専用エンジン版と，元のコードをMicroBlaze上のプログラムとして走行させた際，両者が同一のデータを同一量だけ処理することで，処理の速度を比較した（図20）．

前例の画像フィルタと同様に，シミュレーションで十分に動作検証はできているので，実装には決まった1ブロック・データを1000回繰り返し処理することで，ハードウェアが安定しているかどうかのテストで済ませた．

そのため，図20に適合するようにソフトウェア側に変更を加えた．この変更をシミュレーションで確認した後にCoBuilder

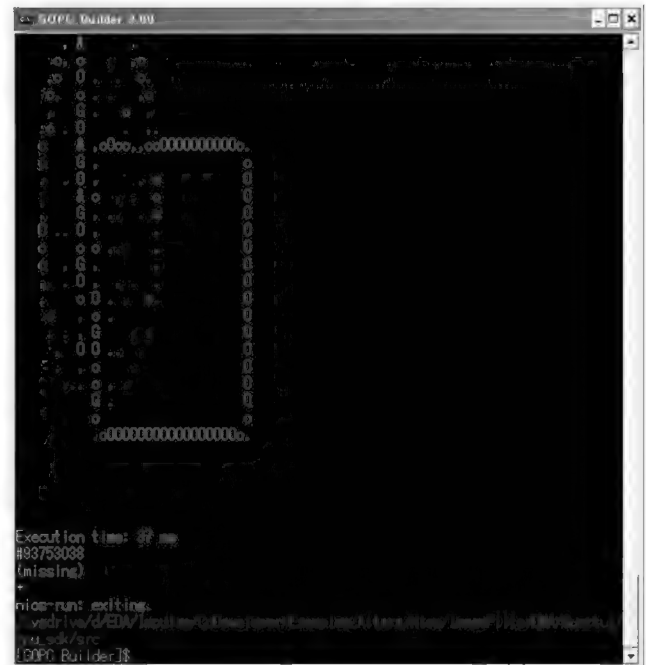


図16　フィルタリング結果の出力

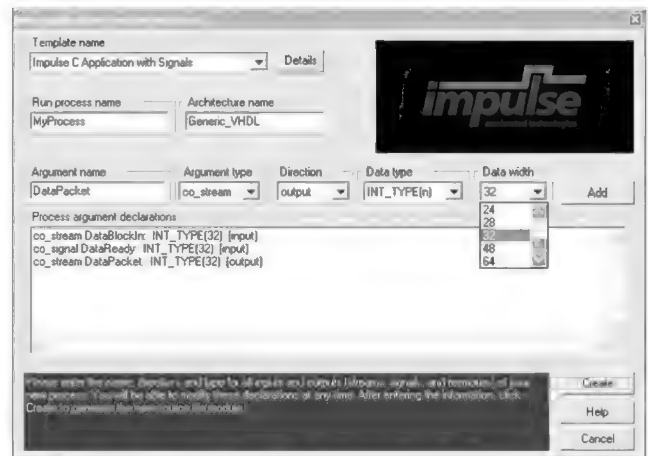


図17　DesignAssistant機能

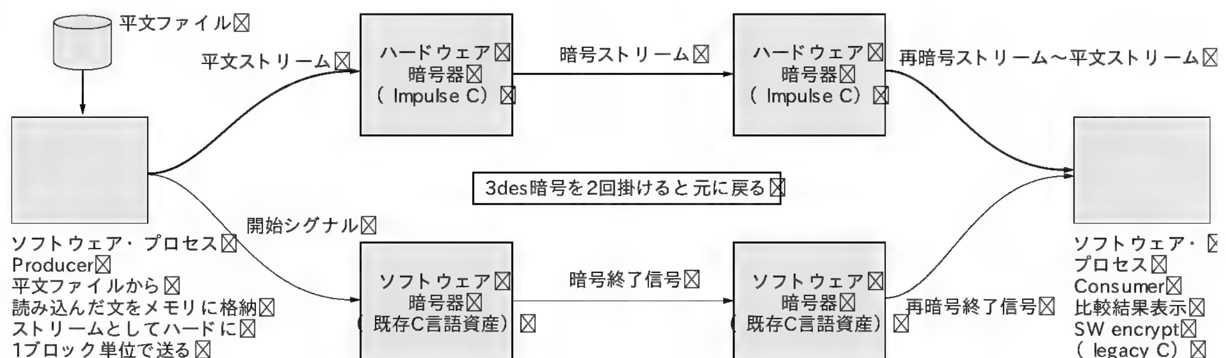


図18　3DES機能検証に使用したシミュレーション・モデル

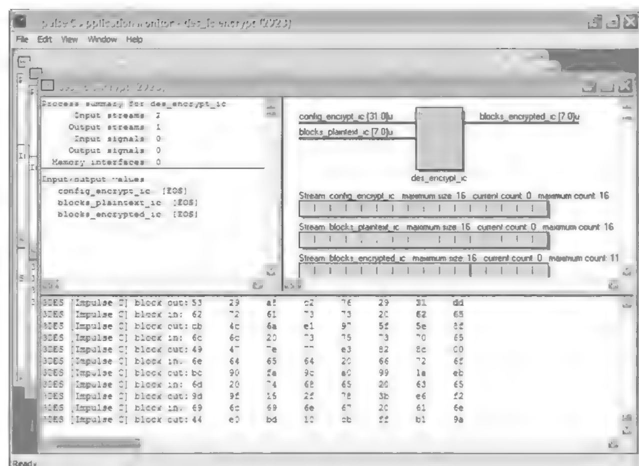


図19　3DES機能シミュレーション

```
Total Number 4 input LUTs: 7,823 out of  10,240   76%
  Number used as logic:              5,972
  Number used as a route-thru:          61
  Number used for Dual Port RAMs:      320
    ( Two LUTs used per Dual Port RAM)
  Number used for 32x1 RAMs:         1,280
    ( Two LUTs used per 32x1 RAM)
  Number used as 16x1 RAMs:              8
  Number used as Shift registers:      182
```

図21　必要としたFPGAのリソース(MicroBlazeを含む)

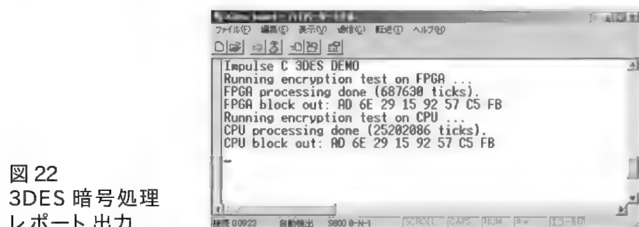
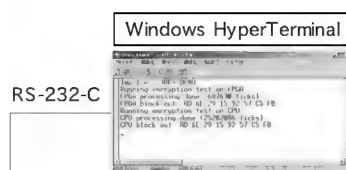


図22
3DES暗号処理
レポート出力



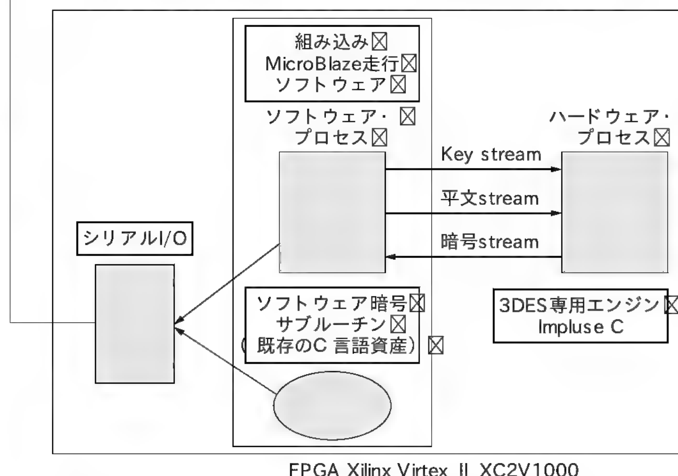


図20　3DES暗号実装検証モデル

に渡した．一般に暗号アルゴリズムはシフト，ビット・スワップ，XORの塊のような構造を持ち，計算量が多く，展開はロジックの爆発ともいうべき大きな回路となる．この3DESの例もCoBuilderにより，180行のC言語ソース・コードが15分程度で約6000行のVHDLに展開された．

ISE上でのコンパイル時間は約90分を要した．使用したVirtex Ⅱ XC2V1000のリソースは，MicroBlazeやそのペリフェラルを含めて図21に示すようになった．

図22は，MicroBlazeがVirtex上に合成された3DES暗号エンジンから終了を受けて，経過時間と受け取った暗号化データを表示し，また自身もソフトウェア版を走らせて経過時間と暗号化の結果を報告したものである．ticksの単位は走行クロック100MHzの周期なので10nsを意味する．

これを見ると，

ハードエンジン版　　6.9 ms
ソフトウェア実行　　252.0 ms

となっており，約37倍ハード・エンジンのほうが速いということがわかる．この時間にはエンジンとCPU間のデータ転送など含まれるので，純粋にはもっと速く，40倍以上であろう．

これは一例だが，既存のCコードをハードウェア化するだけで40倍の高速化が実現できるというのは驚きである．

おわりに

今回，実験に使用したPCの仕様は，Celeron 1.6GHz，メモリ384Mバイトの環境である．

ソフトウェア設計でアセンブラが必要な場面はどのくらいあるのだろうか？　多くのCコンパイラはコンパイル結果のアセンブラ出力をする機能があるが，これを見て手を入れる人はどのくらいいるのだろうか？

現在，これらはかなり特殊なケースとなっていると思う．ハードウェア設計でも似たようなことが起こるのだろうか？

筆者は，案外近いうちに普通になるのではないかと想像する．多分，違う意見の方も大勢いると思う．議論してみたいものである．

PSPのような手段が普通になれば，スーパ・エンジニア1人がそれを用意すれば，ほかの多くの人は抽象の世界で済むはずだ．

くらしげ・かつみ　インターリンク　http://wwwilink.co.jp/

「VxWORKS」を使ったRTOS技術の基礎と応用



第6回

RTOS再入門――排他制御と同期，タスク間通信，タイマ同期

高山 剛

RTOSと組み込みシステム

●「リアルタイム」の意味

まず，リアルタイムOS（以下，RTOS）の「リアルタイム」の意味について誤解が多いので，それを解いておきます．

「リアルタイム」をゲーム・マシンのリアルタイム性や，新幹線の座席のオンライン予約システムのような，人間が何かしたときに，不快を与えない時間内に応答してくれるという意味だと理解している方がいますが，それは違います．RTOSが実現する「リアルタイム」の意味とは，対機械，対デバイスの許容できないms，μs単位の応答の遅れを防ぐため，OSが応答性のワースト・ケースを保証することで，その機械やデバイスの致命的な誤作動を防ぎ，絶対の信頼性を提供することを意味します．

座席の予約システムでは，1～2秒内のレスポンス・タイムを実現できればリアルタイムと呼べるでしょう．逆にロボットの触覚センサからのデータを読み取ったり，アクチュエータを制御する場合にはms，μs単位の応答性が必要です．このとき，ただ速いだけでは不十分です．ロボットは多数のセンサや多数のアクチュエータを同時並行で制御するため，どのような事象が重なってもワースト・ケースでの応答性が保証されていることが重要になってきます．つまり，一定の範囲内での応答性が確実に確保されなければなりません．このことを英語では「deterministic」といいます．日本語では「決定論的」などと訳されています．

●「組み込みシステム」の意味

ここでは，「組み込みシステム」の定義を「コンピュータを応用したインテリジェントな機器のこと」とします．そのため，Windowsや，LinuxなどのOSを用いた機器も，「組み込みシステム」に含まれます．

ここでも誤解を解いておきましょう．組み込みシステムには必ずRTOSが必要だとか，すべての組み込みシステムにはRTOSが適していると理解している方もいるようですが，それは違います．RTOSどころかOSすら必要としない組み込みシステムも存在します．図1にその関係を示します．

OSを用いるようなシステムの場合，製品の製造コストは関係なくその性能と機能を重要視し，リアルタイム性が必要ないことがわかれば，ワークステーションやPCを機器に組み込んで，早期に製品を投入してマーケット・シェアを獲得することは理にかなっています．

しかし，このようなOSでは，次のような要件を満たすことができません．

- クイック・ブート――電源を入れてから画面に映るまでに何秒も待たされるテレビは消費者に受け入れられない
- ディスクレス
- 低消費電力――電圧，周波数を限界まで下げなければならない
- 高コスト・パフォーマンスのプロセッサの選択
- 多種多様なプロセッサへの対応――RISC，CISC，ネットワーク・プロセッサ，マルメディア・プロセッサ，SoCなど
- 限定されたメモリ・サイズ
- 独自ハードウェアへの柔軟な対応――PCと違い，ハードウェアの違いが製品の差別化となる
- 独自のHMI（ヒューマン・マシン・インターフェース）――この違いが製品の差別化となる
- ROM化
- クロス開発環境
- ハードウェアに密着した制御――ドライバのかたまりのような組み込みシステムも存在する
- 高信頼性

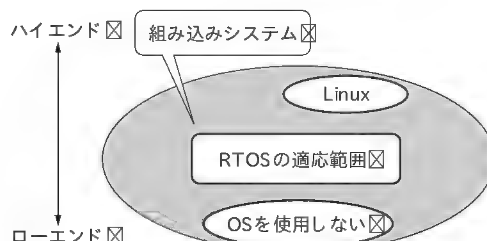


図1 「組み込みシステム＝RTOS採用のシステム」ではない
RTOSを必要としない組み込みシステムも存在する．

● リアルタイム性――高い割り込み応答性，小さな割り込み遅延，deterministic

逆に，リアルタイム性のみに優れたOSであっても，次のような点で弱いOSは将来的な展望を望めず，将来の製品のロード・マップを描けません．

● ネットワーク・プロトコル
● IPv6
● マルチプロセッサ
● GUI/HMI
● ファイル・システム
● コネクティビティ
● メモリ・プロテクション
● トランスペアレント(ファイル・システム，ネットワーク，入出力デバイス，ブロック・デバイス)
● ミドルウェア，ドライバ
● 拡張性
● 情報量
● 長期間のサポート
● ツール

ここで注意しなくてはならない点は，デスクトップ向けやサーバ向けOSでは，後からリアルタイム性を持たせることが困難なことです．最初からリアルタイム性を考慮してOSを設計し，OSのみならず，そのOSに対応するドライバやミドルウェアも含めてリアルタイム性を考慮した設計がなされていなければ，リアルタイム性の実現は困難です．

組み込みシステムでは，低機能なうちはリアルタイム性がなくても支障がないものも多くありますが，製品が高機能化し，複雑化してくると，ある時点でリアルタイム性を要求される場合があります．

そのため，将来を見越して，非RTOSではなく，あえてRTOSを選択するという例もあります．

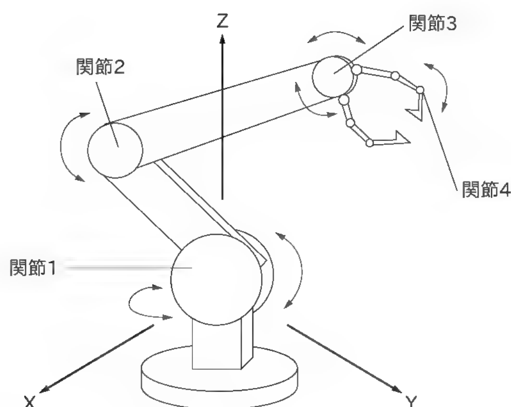


図2 ロボット・アーム
X，Y，Z軸方向の動作をそれぞれ一つのタスクに担当させる

リアルタイム性とマルチタスク

では，RTOSとはどのようなOSで，どのようなしくみなのでしょうか，という説明になると，マルチタスクOS，優先度に基づくスケジューリング，プリエンプティブ…となり，必ずマルチタスクOSが登場します．

本来，リアルタイム性とマルチタスクとは直接関係はないのですが，組み込みシステムとマルチタスクは，非常に相性が良いことが知られています．

たとえば，図2に示すようなロボット・アームの制御では，X，Y，Z軸方向の動作をそれぞれ一つのタスクに担当させます．通信が必要であれば，新たにタスクとして実現し，通信が何チャネルも必要であれば1チャネルごとに一つのタスクを割り当てることで，一つのプログラムで対応することが容易に実現できます．

また，プラント・システムでの気圧や温度は，センサごとにタスクを生成してモニタリングし，各センサからの情報はデータ解析を行うタスクへ送信され，解析や分析，加工といった処理が行われます．さらに，そのデータはHTTPサーバやファイル・サーバを実現するタスクによって，インターネット経由で外部のPCと共有することも可能になります．

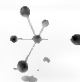
決してリアルタイム・システムにマルチタスクが必要であるわけではないのですが，マルチタスクは，組み込みシステムや実世界に非常にマッチしています．

とくに組み込みシステムでは，複数のタスクが連携しあい，ハードウェアとソフトウェアがシステム全体で一つのアプリケーションとして成り立っています．このあたりがデスクトップOSのアプリケーションとは違うところです．

一般的なデスクトップ・アプリケーションにおいて，シングル・タスクで実装して，ウィンドウ・システム上でまったく別個のアプリケーションとして動作させ，連携協調して一つの目的を遂行するアプリケーションというものはまれでしょう．

マルチタスクを使用せずにシステムを作ると，図3のように大きなループの中で各事象を一つずつ処理していくことになります．事象の数が少なければ，極めて高速に，そしてdeterministicに動作するシステムができあがります．しかし，事象の数が増えて機能が複雑化すると，プログラムは複雑化し，応答性が予測できなくなり，各事象に対する優先順位が付けにくくなるといった問題を抱えることになります．とくにメンテナンス性の悪さや再利用性の低さは問題です．

たとえば，RS-232-Cでシリアル通信を行う場合には，RS-232-Cの初期化コードとループ内で送信処理や受信処理，さらにはソフト・フロー制御のためにXONやXOFFをチェックしたりと，ループの中にRS-232-Cを制御するためのプログラムがあちこちに散在することになり，とても見通しが悪くなります．ネットワーク・プロトコル・スタックなどになると，実装



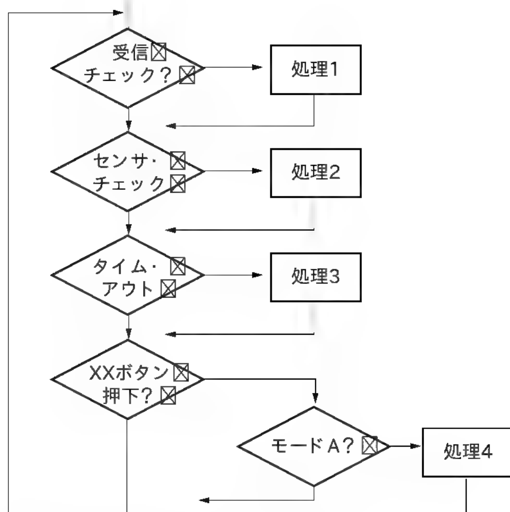


図3　マルチタスクを使用しないシステム
大きなループの中で各事象を一つずつ処理する．

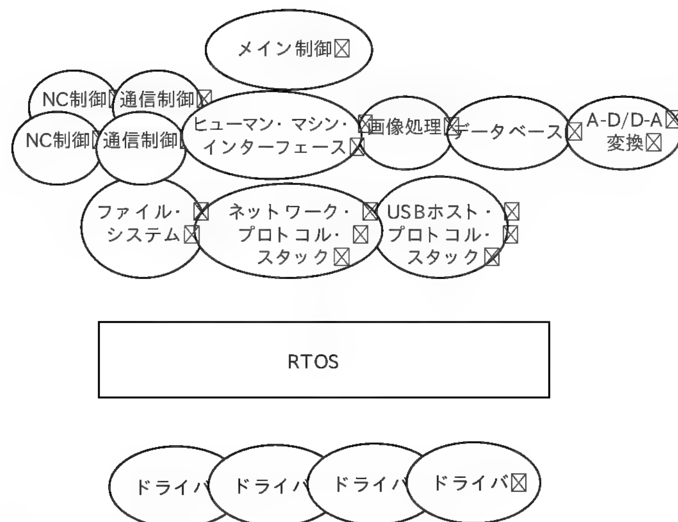


図4　マルチタスクOSでオブジェクト化した例
OSやミドルウェアといった各システム構成要素を一つ一つのタスクに分けることでオブジェクト化した例．システム全体をシンプルに構成でき，メンテナンス性も高まる．

リスト1　二つのタスクA，Bが同一のリソースを操作や参照する関係――やっかいな問題が発生
タスクAは変数CNTを＋1し続け，タスクBは1sに1回変数CNTをゼロ・クリアする．

```
{
  for( ;;)
   {
   cnt++;
   }
}
```

（a）タスクA

```
{
 for( ;;)
  {
  taskDelay( 60) ; /* 60 tickをウェイトする.
                     1tickはデフォルトで1/60s周期 */
  cnt = 0;
  }
}
```

（b）タスクB

はお手上げになってしまいます．

そのような場合，図4のようにマルチタスクにより，各システム構成要素を一つ一つのタスクに分けることでオブジェクト化を進め，その一つ一つをシンプルに構成することで，どのような複雑なシステムもシンプルなオブジェクトの集まりにすることができます．これにより，システム全体をシンプルに実装できるでしょう．

このように，マルチタスクOSを採用すれば，ドライバ，OS，プロトコル・スタック，ミドルウェア，アプリケーションというプログラムの単位でオブジェクト化することで，メンテナンス性が高まります．

タスクに優先順位が付き，OSがプリエンプティブ（つねに優先順位の高いタスクが現れると，優先順位の高いタスクが即座にスケジューリングされる）であれば，重要な仕事が邪魔されることもありません．

ただし，マルチタスクにすることで，タスクごとにコンテキスト（PC，スタック，レジスタ）を持たせ，メモリを余分に必要としたり，スケジューラが必要になるため，メモリ・サイズが大きくなり，OSのオーバヘッドも発生してきます．しかし，RTOSによるdeterministicの特性はこれらのデメリットを相殺するので，メモリを多少余分に消費するという犠牲を払っても，十分にマルチタスクを採用するだけの意味があるわけです．

排他制御の方法

さて，マルチタスクでシステムを構成するとシステムが見通しやすくなり，優先順位を付けることで設計しやすくなることは理解できたと思います．

ただ，マルチタスクを実現した場合，タスクAとBがまったく関係のない仕事を行っている場合は問題ないのですが，非常に密接に同一のリソース（ハードウェア，メモリ）を操作や参照する関係にある場合には，やっかいな問題が発生します．

たとえば，リスト1のようにタスクAは，変数CNTを＋1し続けます．タスクBは1sに1回，変数CNTをゼロ・クリアします．

リスト1は，このタスクAとBとの協調によって，1秒間にタスクAが何回ループできるかを計測できそうに見えますが，実際にプログラムを実行させるとおかしな結果を出すことがあります（異常に大きなカウント数を計測する）．

タスクAのCNT++ですが，プログラムをアセンブラで展開するとRISCプロセッサでは，

```
0010 LD CNT -> REG1
0014 ADD 1+REG1 -> REG1 （レジスタ1に＋1を加算）
0018 ST REG1 ->CNT
```

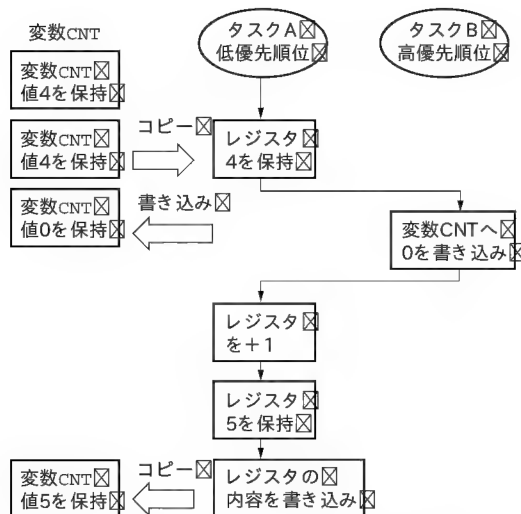


図5　タスクBがゼロクリアされたことが無視される

といったように，1個の命令でなく，複数の命令になります．

仮に，アドレス0014で割り込みが発生し，タスクBがスケジューリングされるような事象が発生し，タスクBがスケジュールされて，変数CNTに対してゼロ・クリアを実行したとします．このとき，タスクAに実行権が戻ると，古い内容を持っているREG1を変数CNTに代入するので，せっかくタスクBがゼロ・クリアしたことが無視されてしまいます(図5)．

0010から0018のようなプログラムを，クリティカル・セクション(きわどい領域)と呼びます．

リスト構造のデータ操作やファイル・システムのFATなど，さまざまなデータ構造の処理でクリティカル・セクションは存在します．

また，FDC，シリアル・コントローラ，フレーム・グラバなどのハードウェアに密接したI/Oについても，一つのI/Oレジスタが複数の役割をもつ場合や，複数のレジスタの操作を合わせて一つの処理が必要な場合に，その一連の処理がクリティカル・セクションとなります．

たとえば，シリアル・コントローラでボーレートを変更する

Column 1

排他制御(その1)

● 割り込みロックによる排他制御

セマフォを使わずに，次のように割り込みロック(禁止)を使って簡単に排他制御を行うことも可能です．

```
oldLockLevel = intLock();
cnt = 0;
intUnlock(oldLockLevel);
```

しかし，この場合，次のような問題点があります．

排他制御は，システム中のOSやアプリケーション，ドライバがクリティカル・セクションに属するデータ構造やハードウェアに適用するため，非常に多くの個所で必要になります．これを一つのクリティカル・セクションごとでなく，割り込みロックのひとくくりで行うと，割り込みレイテンシの増加を引き起こします．

- セマフォの場合は，semTake実行後も割り込み可能で，クリティカル・セクションの処理が長くても，その処理時間がほかのクリティカル・セクションには影響を与えない
- 低優先順位のタスクがintLock()/intUnlock()していると，そのクリティカル・セクションとは関係のない高優先順位のほかのタスクまでが実行できない

このように，割り込みロックで排他制御を行うとRTOSの利点を損なってしまいます．

一方，排他制御セマフォは次のような利点があります．

- 割り込みロックと比較してリソース単位に排他制御が可能
- 待ち行列として，FIFO，TASK PRIORITYを選択できる
- タイム・アウトを指定できる
- タスク・デリートからの保護
- 入れ子が可能(割り込みロックも可能だが，割り込みロック期間が無視できない大きさになってしまう)
- プライオリティ・インヘリタンス(本連載第1回，本誌2003年11月号を参照)

ただし，排他制御セマフォは割り込みサービス・ルーチン(ISR)では使用できません．ISRでは，割り込みロックを排他制御に使用しなくてはなりません．

● 割り込みサービス・ルーチン(ISR)での排他制御

ISRでは，セマフォを使用できません．なぜなら割り込みサービス・ルーチン内で待機させることは，現在のコンピュータ・アーキテクチャ上，許されないからです．

そこで，ISRでは割り込みロックを使用します．ただし，割り込み禁止区間を最小限に留めることが求められます．もし，割り込み禁止期間が長くなる処理が必要になった場合は，タスクへメッセージ・キューで同期を取り，タスク・レベルで処理することで割り込みレイテンシを最低限に留めなければなりません．

また，ISRとタスクが同一のクリティカル・セクションを持つ場合には，割り込みロックを使用することになります．できればISRでクリティカル・セクションを処理することは避け，メッセージ・キューでのタスク同期通信を経て，タスク・レベルで処理し，クリティカル・セクションをセマフォで排他制御するべきです．このとき，どちらの方法を取るかは，アプリケーションのトレードオフを考慮して選択すべき問題です．

RTOSが提供するカーネルやミドルウェア，ドライバは上記のように排他制御をなるべく使うようにして，割り込みロックをできるだけ避けるように設計されています．この点がRTOSと非RTOSの違いでもあります．

場合，シリアル・コントローラの働きを止めて，分周値を変更し，再スタートしてやらなければなりません．この一連の処理もクリティカル・セクションとなります．なぜなら，シリアル・コントローラがストップしているときに，ほかのタスクがシリアル・ポートにデータを送信したら，予測できない事態を引き起こしてしまうからです．

RTOSは，(マルチタスクであれば必ず)クリティカル・セクションに対して排他制御という手続きを踏んで制御を行わなければなりません．排他制御というのは文字どおり，二つ以上のタスクが同時にクリティカル・セクションをアクセスする場合，一つのタスクだけが許され，その処理が完了するまでほかのタスクは待機(VxWORKSでは"PEND"されると表現する)させられます．

VxWORKSをはじめとするRTOSには，排他制御するために次の三つの方法があります．

- 割り込み禁止
- 排他制御セマフォ
- プリエンプティブ禁止

ここでは，セマフォによる排他制御を説明します．割り込み禁止，プリエンプティブ禁止による排他制御の説明は，本文中では割愛し，コラム1とコラム2に示しました．これは，排他制御は基本的に排他制御セマフォで行うということが基本ですが，ハードウェアの特性によって性能を発揮することが困難であったり，割り込みハンドラでI/O処理を扱う必要があったり，ハードウェア的に制限があったりする場合，セマフォ以外の方法を使うことがあるからです．実際にプログラムを書いて，「うーん，何かつごうが悪いな」とわかったときには，コラム1とコラム2を思い出してください．

排他制御セマフォのしくみ

リスト1のプログラムを，セマフォを使ってクリティカル・セクションの部分を排他制御にすると，リスト2に示すプログラムになります．クリティカル・セクションへのアクセスの前後で，セマフォをTakeし[注1]，クリティカル・セクションの処理を終えるとただちにセマフォをGiveし，ほかのタスクがそのクリティカル・セクションにアクセスすることを許します．

セマフォというのは，OSの機能として古くから知られる手法で，「手動信号機」という意味です．信号が"侵入可能"を示せば侵入を許し，"停止"を示せば停止しなければばらず，再び"侵入可能"を示すまで待たねばならないしくみ(モデル)をOSに応用したところから由来しています．

セマフォはC言語の構造体で表現され，構造体のメンバとしてフラグを持ち，'0'と'1'の値を取ります．

リスト2 セマフォを使ってクリティカル・セクションの部分を排他制御に

リスト1のプログラムを，セマフォを使ってクリティカル・セクションの部分を排他制御にした．

```
{
  /*
  排他制御専用セマフォを生成
  SEM_FIFO 待ち行列でFirst In First Out
  ほかにSEM_PRIORITYを指定するとタスクの
  優先順位で待ち行列の優先順位が決定される
  */
  semId = semMCreate( SEM_FIFO) ;

  for( ;;)
   {
   /*
   WAIT_FOREVER 永遠に待ち続ける
   数値を指定すると，その時間( Tick) を超えると
   ERRORを戻り値として返す
   指定の待ち時間が経過してもセマフォを
   獲得できなかったことを知ることができる
   */
   semTake( semId , WAIT_FOREVER) ;
   cnt++;
   semGive( semId) ;
   }
}
```

(a) タスクA

```
{
  for( ;;)
   {
   taskDelay( 60) ;  /* 60 tickをウェイトする
                       1tickはデフォルトで1/60秒周期 */
   semTake( semId,WAIT_FOREVER) ;
   cnt = 0;
   semGive( semId) ;
   }
}
```

(b) タスクB

リスト2のプログラムでは，semTake()でフラグをチェックして，'1'であれば，OSはそのままプログラムの実行を許します．'0'であれば，OSは'1'になるまでタスクを待機(PEND)させます．semGive()は，フラグを'1'にすると同時に，OSはそのセマフォを待っているタスクがあればそのタスクを実行可能な状態(READY状態)にするわけです．

このセマフォを使ってハードウェアの制御レジスタをプログラムする際，クリティカル・セクションがあった場合には，そのクリティカル・セクションをアクセスするときに，すべてのプログラムがセマフォを使ってアクセスしなければなりません．

クリティカル・セクションにおいて，ハードディスクやフロッピ・ドライブはドライバが必要な排他制御を行っています．各ファイルへのread/writeについてもOSが内部データ構造を排他制御します．したがって，ハードディスクやフロッピ・ドライブ上のファイルについてread()やwrite()を実行する場合には，排他制御には必要ありません．

しかし，アプリケーションやドライバで共有するデータや，

注1：多くのOSの書籍では"取得"や"フルにする"などの表現が使われているが，ここではVxWORKSの関数がsemTake/semGiveなので，いちいち日本語→英語→関数名という変換を頭の中で行うことを避けるため，Takeする/Giveするという表現を使う．

Column2
排他制御（その2）

● タスク・プリエンプティブ禁止による排他制御

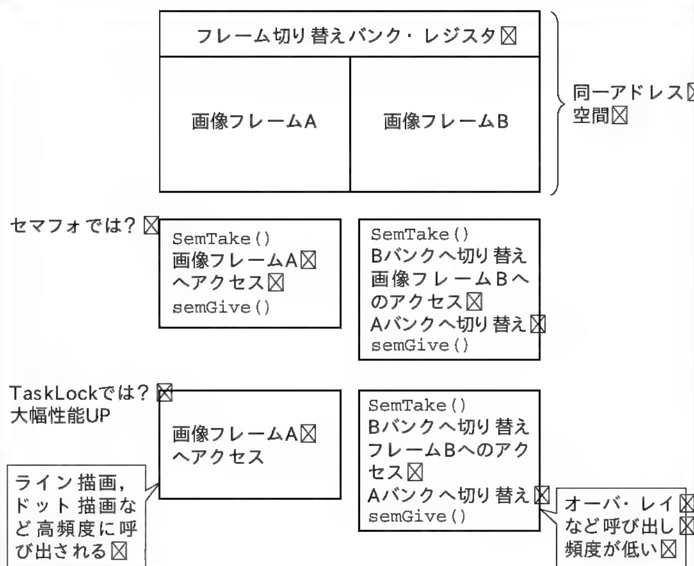
第3の排他制御として，タスク・プリエンプティブ禁止〔taskLock()/taskUnlock()〕による排他制御も提供されています．taskLock()/taskUnlock()で囲まれた区間は，仮に高い優先順位のタスクが現れても，タスクのスケジュールが行われません．つまり，taskLock()/taskUnlock()区間では，ほかのタスクが動作することはあり得ないことを意味します．すなわち，排他制御が実現されていることになるのです．

タスクからの排他制御は，コラム1と同じ理由でtaskLock()よりセマフォを使用すべきであり，ISRからはtaskLock()を使用できないので，割り込みロックが適切です．一見，このtaskLock()によるタスク・プリエンプティブ禁止の存在価値はないように思われますが，アプリケーションの性質によってはtaskLock()がもっともふさわしい場合もあります．

たとえば，図Aのように画像メモリがあって，バンク切り替えを行ってアクセスしないと2枚のフレーム（AとB）がアクセスできないボードがあるとします．

セマフォを使うと，AをアクセスするときもBをアクセスするときも，同じクリティカル・セクションなので，AへのアクセスもBへのアクセスも必ずセマフォを使った排他制御が必要です．一方，taskLock()を使うと，Bへのアクセスだけ（Aだけの場合も同様）taskLock()を使うと，排他制御が可能です．単純に考えると，OSのシステム・コールが半分になるので，画像処理のパフォーマンスの向上に大いに貢献します．

この画像メモリAとBがそれぞれRGB画像やオーバ・レイ画像と種類が違うもので，アクセス頻度が大きく違う場合は，頻度の低いほうにtaskLock()を適用することで大幅にオーバヘッドを軽減できるでしょう．



図A　タスク・プリエンプティブ禁止による排他制御の例

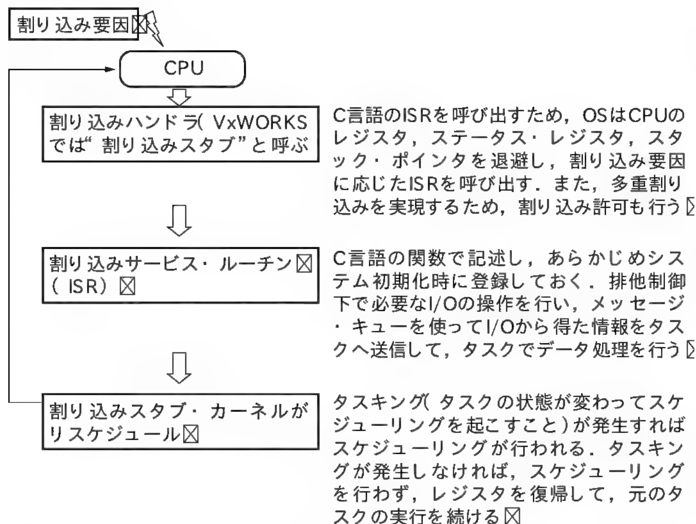


図6　割り込み
割り込みハンドラ部分はOSが担当．割り込みハンドラからアプリケーションが指定したC言語の割り込みサービス・ルーチンが呼び出される．

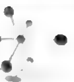
ハードウェアに直接アクセスする場合には，セマフォを使用しなければなりません．

割り込みハンドラ

組み込みシステムのアプリケーションを構築すると，外部との事象と連携した動作が求められます．たとえばディジタル・カメラのシャッタが押された場合，通信ポートからデータを受信するなど，CPUの外で起こる事象（イベント）に対して，それに応答するプログラムを呼び出す必要が生じます．

この場合は，外部からのイベントは割り込みによって通知されることが望ましいとされています．割り込みでなく，タスクがI/Oをポーリングしても同等の機能を果たしますが，CPUのむだ使い，しいてはバッテリのむだ使いとなるうえに，見かけ上のマルチタスクが実現できません．

RTOSでは，割り込みを受けると，OSを介在して割り込み要因に応じた割り込みサービス・ルーチン（ISR）が呼び出されます．CPUから見た場合，割り込みを受け付けて，割り込み



ハンドラ・ルーチンを呼び出します．このとき，割り込みハンドラは，CPUそれぞれに依存したプログラム上の制限が課せられ，アセンブラでの記述が要求されます．RTOSでは，図6のように割り込みハンドラ部分はOSが担当し，OSの割り込みハンドラからアプリケーションが指定したC言語の割り込みサービス・ルーチンが呼び出されます．この実装により，すべての割り込み処理をC言語で記述することができ，CPUに依存しない移植性に優れた割り込み処理が，しいては移植性の高いドライバやアプリケーションが実現できるわけです．

ISRではまず，もっともリアルタイム性が問われるI/O処理やデータ処理を行い，比較的リアルタイム性の低い処理は，なるべくタスクに仕事を任せます．このようにシステムを設計することで，高い割り込み応答が実現できるわけです．

VxWORKSでは，ISRを登録するのは簡単でintConnect(割り込みベクタ，ISRの関数エントリ・アドレス，ISRへの引き数)を実行するだけです．なぜISRが引き数をとるのかと不思議に思うかもしれませんが，一つのシリアル・ドライバで同じ種類の周辺デバイスを複数コントロールする場合を考えてもらえばわかりやすいと思います．引き数にポート番号を指定しておけば，ISRが1個の関数でも，引き数でポート番号を得られるので，複数のポートに簡単に対応できるのです．



同期はセマフォで実現

リアルタイム性が問われるI/O処理をISRで処理し，比較的リアルタイム性の低い処理はタスクに任せると述べました．このためにはISRから通信を行うことになります．この場合，メッセージ通信(VxWORKSの場合はメッセージ・キュー)を行います．

また，データを受け渡す必要がなく，単に特定の割り込みに発生に対して特定のタスクを動作させたい場合，これを同期(Synchronization)と呼び，セマフォで実現可能です．VxWORKSでは同期の目的にバイナリ・セマフォを用います．

メッセージ・キューとバイナリ・セマフォのプログラム例を次に示します．

▶ メッセージ・キュー

```
初期化コード
MSG_Q_ID qID;
qID = msgQCreate( QUE_LENGTH, MSG_LENGTH,
                                        MSG_Q_FIFO);
割り込みサービス・ルーチン
{
old = intLock();
I/O処理
data = I/Oからのデータ読み込み
IntUnlock(old)
msgqSend(qID , &data
           ,sizeof(data),NO_WAIT,MSG_PRI_NORMAL)
}

待機しているタスク側
{
msgqReceive(qId,&data,sizeof(data),
                                    WAIT_FOREVER)
データ処理
}
```

※各引き数の意味

QUE_LENGTH ：キューの長さを決める

MSG_LENGTH ：メッセージ長を決める

MSG_Q_FIFO ：メッセージがない，またはキューが満杯のとき，タスクはPEND状態になるが，複数タスクがPEND状態のとき，どれを優先させるかのアルゴリズムを決める．FIFOとタスク・プライオリティを選択できる

NO_WAIT ：メッセージが到着するまで無制限に待つ．タイム・アウト時間も設定できる

▶ バイナリ・セマフォ

```
初期化コード
SEM_ID semID;
semID = semBCreate(SEM_Q_FIFO, SEM_EMPTY);

割り込みサービス・ルーチン
{
old = intLock();
I/O処理
IntUnlock(old)
semGive (semID);
}
```

▶ 待機しているタスク側

```
{
semTake (semID, WAIT_FOREVER);
I/O処理
}
```

以上は，割り込みとタスクの同期，そしてデータ通信についてでしたが，もちろんタスク間でも使用できます．メッセージ・キューにより，ネットワークで市民権を得たクライアント-サーバ・モデルをシングルCPU内で実現できます．クライアント-サーバのモデルは，シングルCPU内でもマルチCPUでも，組み込みシステムではごく自然に使用されています．

タスク分割によるオブジェクト化を進めていくと，必ずクライアント-サーバ・モデルに行き着きます．シングルCPUでもクライアント-サーバ・モデルを使うことはRTOSやUNIXを使うエンジニアにとっては当たり前だと思いますが，OSを使ったことのないエンジニアにとっては驚きかもしれません．

Column3

次世代VxWORKS「VxWORKS 6.0」が登場

「Base6」というプロジェクト名で開発されていたVxWORKS 6.0が発表されました．

メモリ・プロテクションとプロセス・モデルを採用しながら，従来のVxWORKSの特徴を100％継承したOSとなっているようです．4月からPreRelease版が一般に公開される予定になっているので，本連載でも詳しい情報を紹介したいと思います．MMUlessもサポートされるので，この場合の性能低下はゼロですが，メモリ・プロテクションを導入した場合，どれだけのパフォーマンスが出るか気になるところです．

タイマとの同期

「おや，なぜまた同期が登場？」と思われるかもしれません．タイマとの同期も，タイマを必要とするタスク分だけタイマが存在すれば，前述のバイナリ・セマフォで事が足ります．

しかし，タイマをたくさん載せて機能させると，それぞれが割り込みをいっぱい発生させてOSのオーバヘッドが大きくなります．そのうえ，ハードウェアのコストも大きくなります．

そこで，タイマを1個だけ実装して，すべてのタスクが共有すれば問題は解決します．リアルタイム性や，タイマの精度が要求されるのであれば，タイマを必要とするタスクの分だけ実装してバイナリ・セマフォを使ってもかまいません．組み込みシステムの世界は，ハードウェアとソフトウェア間でのトレードオフや，性能とコスト間のトレードオフで合理性があれば何でもありだからです．

話を戻して，タイマとの同期ですが，つまり1個のタイマ（VxWORKSではシステム・タイマと呼ぶ）をいかにして複数のタスクで共有するか，どういったAPIを使うかということになります．具体的に，VxWORKSでは次のようなAPIがあります（ほかにもPOSIX timerが存在する）．

▶**taskDelay()**――タスク・レベルで，タスクを指定tick分だけ遅延させる

一定時間単位でポーリングする場合に有用です．

▶ウォッチ・ドッグ・タイマ――指定tick後にウォッチ・ドッグ・ハンドラを呼び出す

一定時間後にタイム・アウト処理を行う際に有用です．

OSは，システム・クロックの割り込みをいったんOSで管理して，上記二つのAPIをアプリケーションに提供しています．

システム・クロックが一定周期で割り込みを起こすたびにOSがその数をカウントしています．tickは，その数値に当たります．

VxWORKSでは，この値はデフォルトで60tick/sになり，`TaskDelay(60)`は1秒間の遅延を意味しています．もちろん，デフォルトの60tick/sは任意の周期に変更できます．しかし，周期をあまりにも高くしすぎて，OSのオーバヘッドが100%近くなってしまっては，アプリケーションの動作する余地がありません．オーバヘッドが具体的に何%であれば正常かということはいえません．アプリケーションに応じて，高周期によるOSのオーバヘッドと高周期はトレードオフの関係にあり，アプリケーション依存といえます．

*　　　*

ここまで，一般的なRTOSの概論を述べてきました．VxWORKSのアプリケーションは，ANSIやPOSIXを採用し，UNIXライクなAPIを持っているので，UNIXエンジニアであれば本稿で述べた排他制御や同期，タスク間通信，タイマ同期を用いれば，シンプルなシステム向けのリアルタイム・アプリケーションを書くことができるでしょう．しかしながら，完全なリアルタイム・システムを構築するためには，さらに踏み込んで経験を積むことが必要です．

次回は，排他制御，同期，タスク間通信，タイマ同期を用いた典型的なリアルタイム・アプリケーション例を紹介します．ボードへの移植，開発環境，実際の製品にROM化するまでを簡単に紹介して，組み込み開発の実像を紹介します．

たかやま・たけし　ウィンドリバー（株）
takeshi.takayama@windriver.com

PCIと親和性の高いバス規格
StarFabricの技術概要と適用例

◆白井 野之◆

現在，家庭用コンピュータから組み込み機器まで，PCIバスは多くの産業機器に使用されている．

たとえVMEバスのシステムを使用していたとしても，システム制御に使用されるCPU基板内では，ほとんどの場合，PCIバスが使用されている．このPCIバスには複数のI/Oデバイスが接続され，ときにはデバイス間で大容量のデータを転送するためにも使用される．

現在の産業機器からは切り離すことのできないPCIバスであるが，システムのパフォーマンスが上がってきている昨今では，さまざまな問題が浮き彫りになっている．

PCIバスはシェアード・バスであるため，あるデバイスがバスを使用している間は，ほかのデバイスはバスを使用することができない．そのため，複数のデバイスを接続している場合，実効的な転送速度が低下してしまう．

また，PCIバスに接続しているデバイスが故障した場合，PCIバス全体の動作が停止してしまう可能性がある．そのほかにも，接続可能なデバイス数やバスの配線長など，さまざまな制約も存在している．

● PCIバスの問題を解決するスイッチト・ファブリック

スイッチト・ファブリックは，複数の送受信ポートを備えたスイッチを介して，エンド・ノード間をポイント・ツー・ポイント接続する構成をとっている．これにより，

1) デバイス間のデータ転送速度を個々に確保できる
2) デバイスを増やしてもデータ転送速度が低下しない
3) ある通信経路が故障したとしても，ほかの経路に影響を与えない

といった，PCIバスにはない機能を実現することができる(表1)．

StarFabicは，このスイッチト・ファブリック技術を基盤とした規格であり，かつPCIバスとの親和性が考慮されているため，現在のPCIバス資産をそのままにスイッチト・ファブリックを使用できるようになる．

StarFabric規格の概要



StarFabricは，StarFabric Trade Associationで策定された規格である．このStarFabric Trade Associationは，米国のStarGen社が中心となって活動しており，ボード・ベンダが参加している．以下，StarFabric規格の概要を紹介する．

● StarFabricの基本構成

StarFabricの基本的な構成を図1に示す．スイッチト・ファブリックと同様に，ファブリックを構成するためのスイッチと，エンド・ノードとなるブリッジ，各ノード間を接続するリンクで構成される．

スイッチはその名のとおり，ファブリック内を流れるフレーム・データをスイッチする機能をもち，同時に複数のポートを動作させることができる．また，最大7段まで接続することができる．

ブリッジは，エンド・ノードに接続されるバスとStarFabricを接続(ブリッジ)する機能をもつ．

リンクは，送/受信ともに622MbpsのLDVSのライン4対で構成され，最大2.5Gbpsの全二重通信を行う．

各ノード間はCAT5eのUTPケーブル2本で接続され，最大12mまで延ばすことができる．

また，図2のようにスイッチを二重化することで，一方のスイッチに障害が発生しても，もう一方のスイッチでトラフィックを継続することも可能である．

● プロトコル・レイア

StarFabricのプロトコル・レイアを図3に示す．StarFabricのプロトコルは，そのほとんどがハードウェアで制御される．プロトコル制御をハードウェアで行うため，Ethernetのようにプロトコル制御をソフトウェアで行うものにくらべて，CPUの

表1 PCIバスとStarFabricの比較

	PCIバス	StarFabric
接続デバイス数	マルチドロップ型のため，信号品質などの問題で接続デバイス数が制限される	スイッチを介して多数のデバイスを接続可能．スイッチは最大7段まで接続可能
データ転送	データ転送中はデバイスがバスを占有する	ポイント・ツー・ポイント接続によりバスの共有から解放され，複数のデバイスが同時にデータ転送可能
拡張性	物理的なスロット数により制限される	デバイス間を12mまで離せるため，シャーシ間でのバス拡張も可能
障害耐性	デバイスの故障がバス全体を停止させる可能性がある	ポイント・ツー・ポイント接続のため，どこかのデバイスが故障してもほかのデバイスの動作には影響しない．ファブリックの二重化も可能

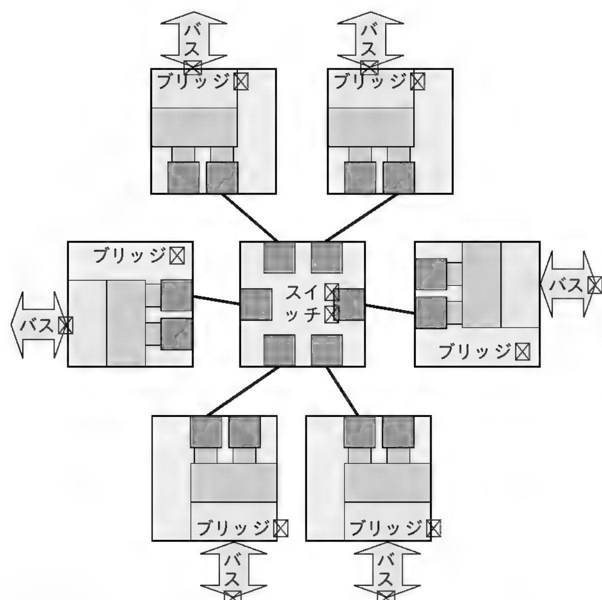


図1　StarFabricの基本構成

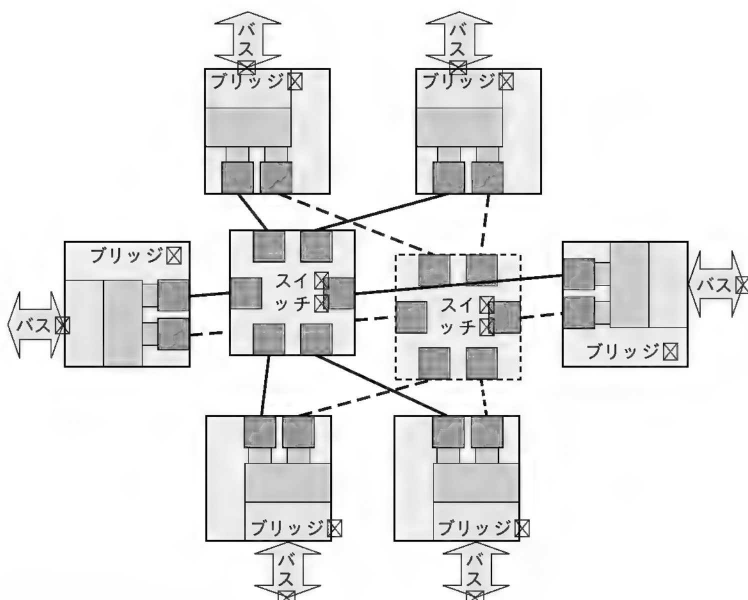


図2　二重化したStarFabric

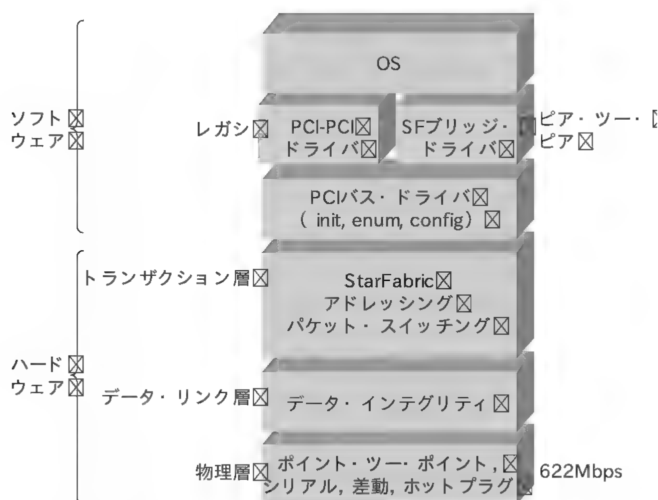
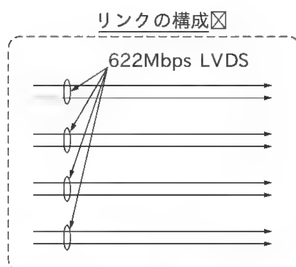


図3　StarFabricのプロトコル・レイア



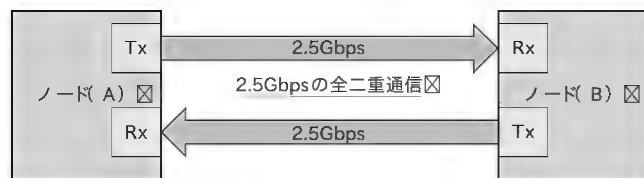


図4　StarFabricの物理層

負荷を低減することができる．

ここでは，アーキテクチャの中心となる物理層，データ・リンク層，トランザクション層について簡単に解説する．

● 物理層

StarFabricは，送信と受信ともに622MbpsのLVDS信号を4対使用して，2.5Gbpsの全二重通信を行う（図4）．622Mbpsと比較的遅いLVDS信号を使用することで，ハードウェアへの実装を容易にしている．また，CAT5eのUTPケーブルのようなシールドのない安価なケーブルでも，通信することができる．

さらに，通常は4対のLVDS信号を使用して通信を行うが，障害などによって4対のLVDS信号のうち3対までが切断されたとしても，通信を継続することができる．これをFragile Link機能という（図5）．

また，ノード間の接続は活線挿抜が可能である．

● データ・リンク層

StarFabricはフレーム単位で通信が行われる．フレームは，ノード間の制御情報とフレームのCRC情報を含むリンク・オーバヘッド，フレームの行き先情報を含むヘッダ，最大128バイトの可変長ペイロードで構成される（図6）．また，フレームは8B/10Bコードに変換されてから送信される．これらを考慮すると，実効的なデータ転送速度は，最大1.77Gbpsとなる．

●トランザクション層

トランザクション層のおもな機能は，フレームのルーティング，フロー制御，エラー制御，イベントなどである．

◆ルーティング

StarFabricでは，フレームのルーティング方法として，アドレス・ルーティング，パス・ルーティング，マルチキャスト・ルーティングの3種類の方法が規定されている．

1) アドレス・ルーティング

PCIバス・システムとの互換性を保つために設けられたルーティング方法である．ルーティング情報として物理アドレス情報を使用し，各エンド・ノードに割り当てられているアドレス範囲を見て，フレームのルーティングを行う(図7)．

2) パス・ルーティング

データの送り先となるエンド・ノードまでの経路(パス)情報を使用して，フレームをルーティングする方法である(図8)．エンド・ノードまでの経路が複数存在する場合，特定の経路にトラフィックが集中しないように設定する．ある経路に障害が発生したときに別の経路に切り替えることなどが可能になる．

3) マルチキャスト・ルーティング

一つのデータを複数のノードに対して転送する際に使用するルーティング方法である．スイッチ・デバイスに事前に転送先となるノードを設定しておき，スイッチでフレームを複製し，各ノードにフレームを転送する．

◆フロー制御

StarFabricでは，クレジット・ベースのフロー制御を行っている．クレジット・ベースのフロー制御では，データを送信する側が，データを受信する側のバッファ容量を把握しているため，受信側のバッファに空きがあるときにのみ，データ送信が行われる．

これにより，むだなトラフィックが発生せず，効率的にデータ通信を行うことができる．

◆トラフィック・クラス(QoS)

StarFabricでは，QoSを実現するために，トラフィックを四つのクラスに分けている．クラスには，優先順位の低いものから，「非同期」，「アイソクロナス」，「マルチキャスト」，「プロビジョニング」がある．

非同期は，一般的なデータの転送に使われる．アイソクロナスは，オーディオやビデオなどのストリーミング・データの転送に使う．マルチキャストは前述のように，複数の宛て先に同じデータを送信するために使う．プロビジョニングは，割り込み信号やStarFabricの制御信号の転送用である．

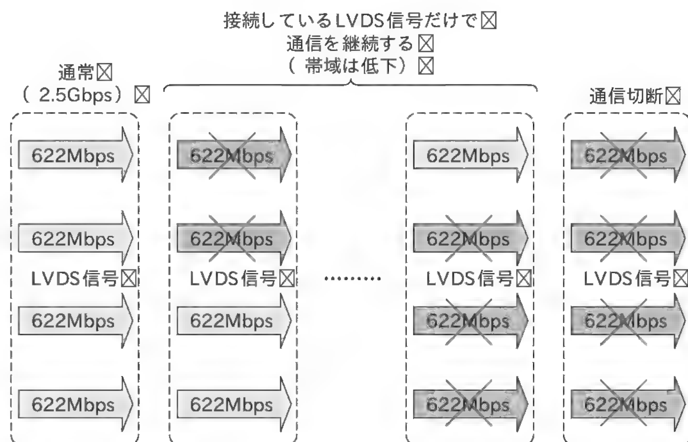


図5　Fragile Link機能

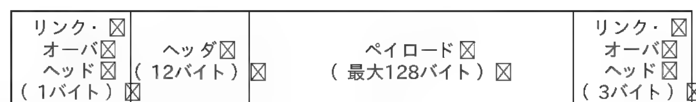


図6　フレーム・フォーマット

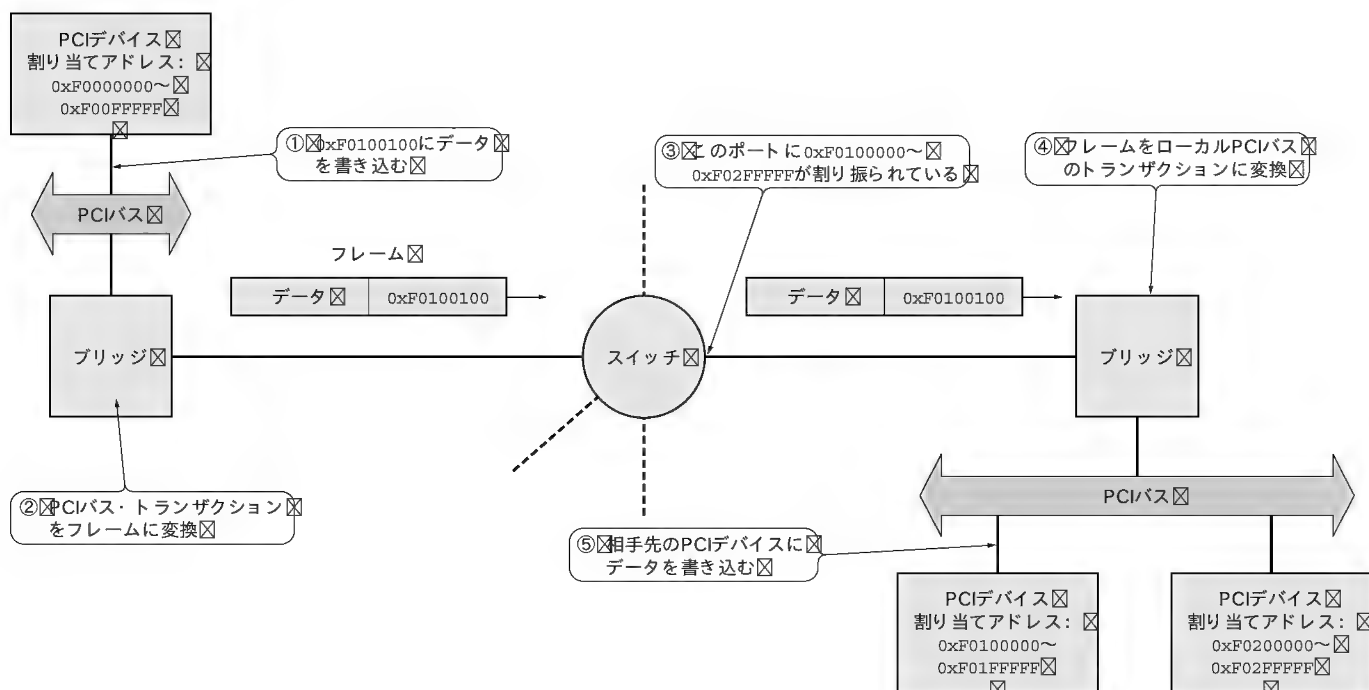


図7　アドレス・ルーティングの例

図8　パス・ルーティングの例

図9　StarFabricのトポロジ

StarFabricのデバイスは，トラフィック・クラスごとにバッファを持っているため，フロー制御はトラフィック・クラスごとに行われる．

◆エラー制御

StarFabricのフレームにはCRCが付加され，さらに8B/10Bの符号化が行われている．データを受信したデバイスでは，8B/10BコードとCRCを確認して，エラーがなければバス上にトラフィックを発生させる．エラーが発生した場合には，フレームの再送処理を行うが，これらはすべてデバイスがハードウェアで自動的に行う．

◆イベント

StarFabricでは，4種類のイベントが定義されており，このイベントを使用して各デバイスの状態やエラー状況を知ることができる．

1) シグナル・イベント

エンド・ノードに接続されているPCIバスの割り込みやエラー情報の通知に使用する．

2) チップ・イベント

デバイス内で発生したエラー情報や，メッセージ情報の通知に使用する．

3) パス・イベント

通信経路状態の通知に使用する．

4) ルーティング・イベント

フレームのルーティング状態の通知に使用する．

StarFabric規格の特徴



StarFabric規格は，すでに幅広く使用されているPCIバスの資産をむだにしないよう，PCIバスとの親和性を重視している．

特にブリッジには，StarFabricをPCI-PCIブリッジとして機能させるためのレガシ・ブリッジ機能と，異なるPCIバス・セグメントをブリッジするゲートウェイ機能を備えている．

● レガシ・ブリッジ機能

StarFabricの大きな特徴の一つに，PCI-PCIブリッジとして動作させるためのレガシ・ブリッジ機能がある．この機能は，StarFabricのアドレス・ルーティングを使用するもので，標準のPCI-PCIブリッジと100%の互換性をもっている．

この機能を使用して，図9のようなトポロジで接続した場合，

図10　ソフトウェアからみたバス構成

物理的にはStarFabricで接続されているものの，ソフトウェアからみると図10（p.143）に示すようなPCIバスとして認識される．すべてのStarFabricデバイスは，PCI-PCIブリッジとして機能するため，BIOSやOS，デバイス・ドライバといったソフトウェアを変更する必要がない．

図11に，レガシ・ブリッジ機能による接続例を示す．

具体的には，Windows 2000の動作しているPCにStarFabricブリッジ・デバイスを載せている．

また，別シャーシには，StarFabricブリッジとSCSIデバイスを載せており，PC側のブリッジと接続をしている．このとき，Windowsのデバイス・マネージャをみると，StarFabricが標準PCI-PCIブリッジとして認識されていることがわかる（図12）．

● ゲートウェイ機能

異なるPCIバス・セグメントをもつ複数のプロセッサ・ボードをStarFabricに接続する場合，レガシ・ブリッジ機能で接続することはできない（異なるバス・セグメントであるため，同

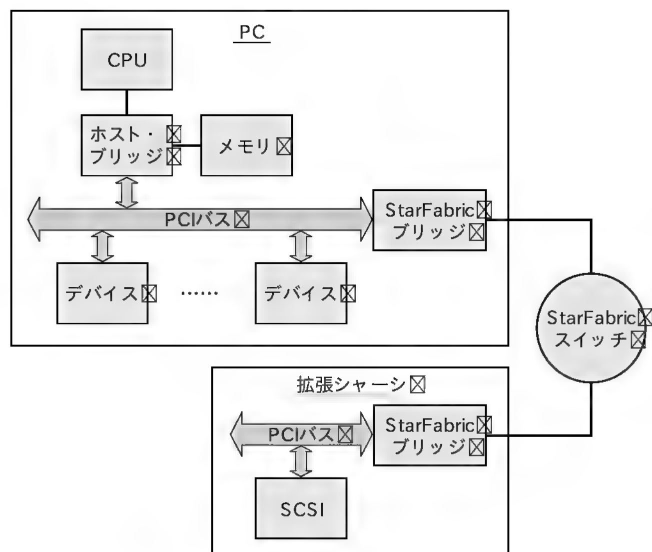


図11　レガシ・ブリッジによる接続例

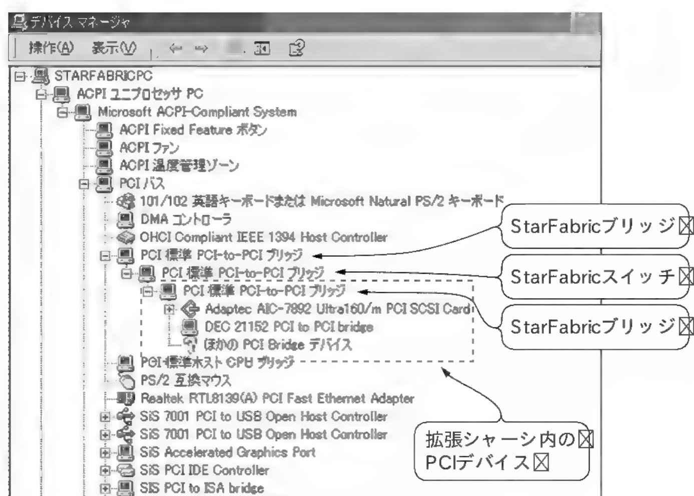


図12　Windowsにおけるレガシ・ブリッジの認識

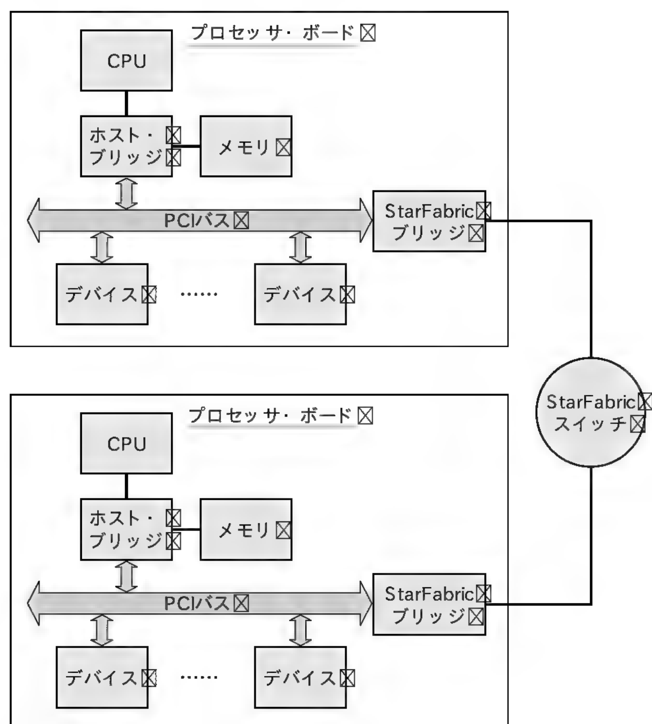


図13　ゲートウェイ機能による接続例

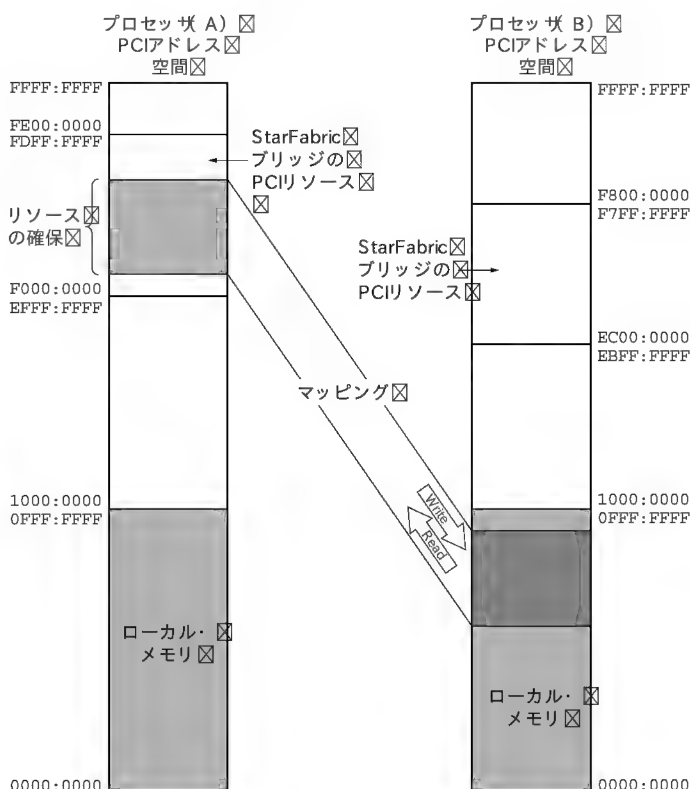


図14　マッピング

じアドレスが複数存在してしまい，アドレス情報によるルーティングができない）．

ゲートウェイ機能は，パス・ルーティングを使用するため，StarFabricの中ではアドレス情報を使用しない．そのため，複数の異なるPCIバス・セグメントを接続することが可能となる．このとき，ゲートウェイ機能は，非トランスペアレントのPCIブリッジのような役割をする．

ゲートウェイ機能により，複数のプロセッサ・ボードを接続したときの例を図13に示す．各プロセッサ・ボードは，それぞれがPCIバスをもっており，そのPCIバス上にローカル・メモリとStarFabricのブリッジ・デバイスが存在している．

各プロセッサ・ボード間でデータ通信を行う場合，各プロセッサ・ボードのPCIバス上に，通信先であるプロセッサ・ボードのPCIバスの一部をマッピングして，そのマッピングされた領域を通して通信を行う．

ここで，プロセッサ（A）が，プロセッサ（B）のローカル・メモリに対してデータ転送を行う場合について考える（図14）．このとき，プロセッサ（A）はプロセッサ（B）のローカル・メモリにアクセスするために，プロセッサ（A）のPCIバス上にリソースを確保する．このリソースと，プロセッサ（B）側のローカル・メモリのアドレスをマッピングすることで，プロセッサ（A）は，プロセッサ（B）のローカル・メモリに対して，リソースを通してアクセスすることが可能になる．

● アプリケーション例

◆ PCIバス拡張

前述のとおり，StarFabricはPCI-PCIブリッジの機能をもっている．この機能を使用することで，PCIバスの拡張を容易に実現することができる．

一例として，ホスト・システムとなるPCと，センサとなるA-D変換基板を接続している単純なシステムで考えてみる．

ここで，アナログ信号の品質を保つために制御端末となるPCと，センサであるA-D変換基板は，できるだけ離して配置することとし，別シャーシでの構成とする．

従来は，図15のようにA-D変換基板をPCIバス拡張シャーシに組み込んで，専用ケーブルを用いてPCと接続していた．しかし，この方法ではパラレル・バスの信号をそのままケーブルで延長するためケーブル長を長くすることが難しく，1～2m程度が限界であり，機材の配置に自由度がなかった．

このシステムにStarFabricを使用した場合の構成が図16である．PCI-PCIブリッジ機能を使用することで，PCと拡張シャーシは同一のPCIバスとして接続される．BIOSやOSなどのソフトウェアからは，StarFabricの部分が「標準PCI-PCIブリッジ」として認識されるため，従来のソフトウェアを変更せずにそのまま使用することができる．さらに，PCと拡張シャーシ間はCAT5eのUTPケーブルを使用して最大12mまで離すこともできる．

また，図17のようにPCと拡張シャーシ間にスイッチを入れることで，ほかの拡張シャーシをさらに追加することもでき，拡張性を広げることができる．

◆プロセッサ間通信

従来からプロセッサ間通信には，比較的容易に使用できることから共有メモリ・タイプのインターフェースが利用されてき

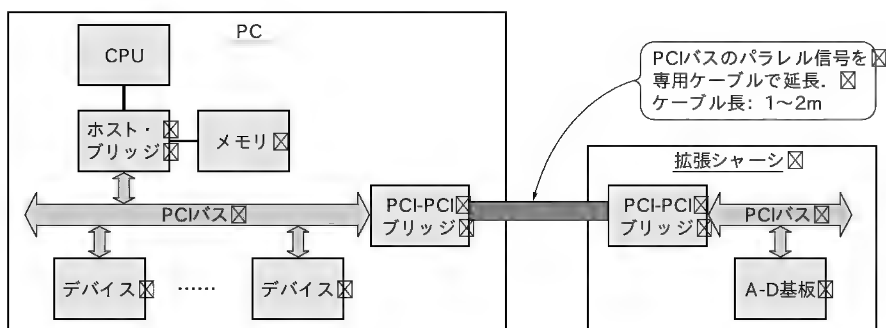


図15　従来のPCIバス拡張

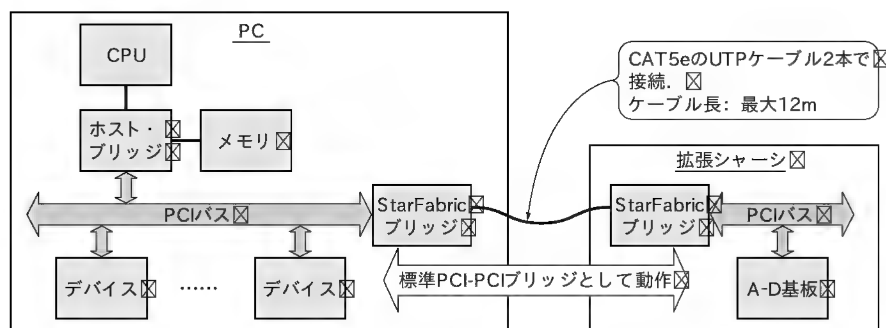


図16　StarFabricを使用したバス拡張

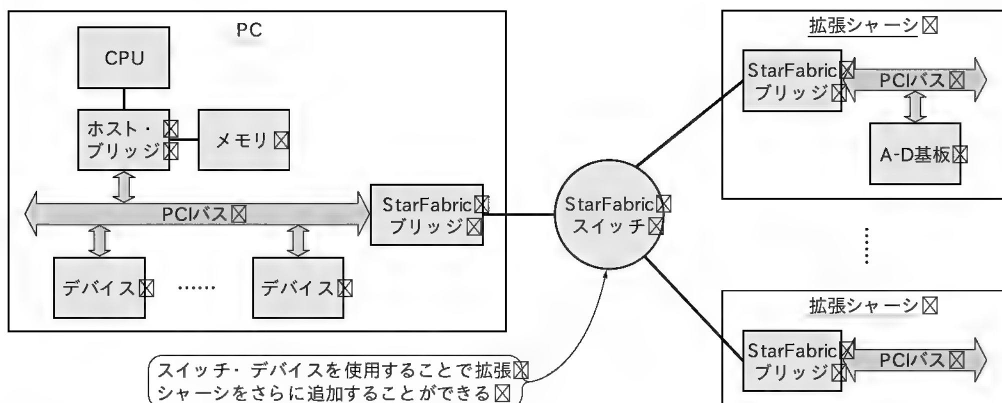


図17　スイッチ・デバイスを使用したPCIバス拡張

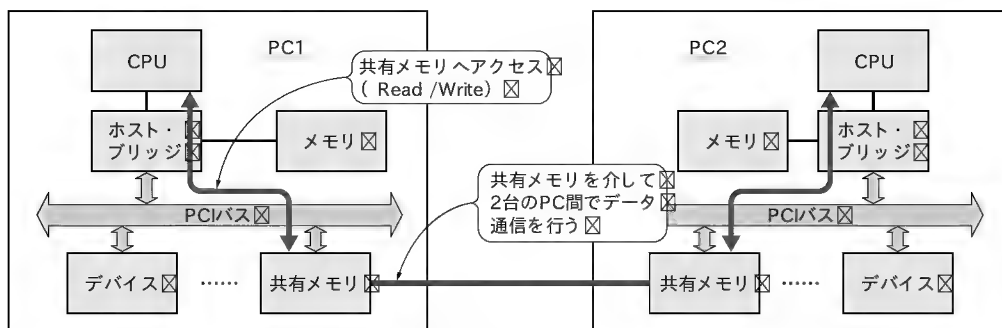


図18　共有メモリ・インターフェースによるプロセッサ間通信

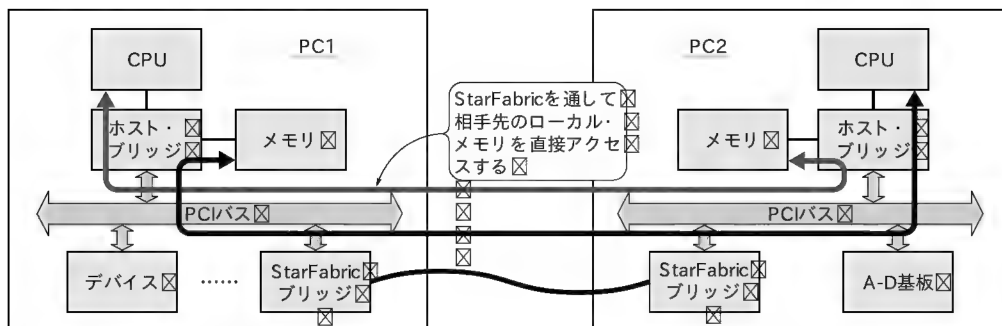


図19　StarFabricによるプロセッサ間通信

た（図18）．この場合，各プロセッサ・ノード間でやりとりするデータは，すべて共有メモリに書き込まれる．したがって，CPUがデータを取得するにはPCIバスを経由する必要がある．

これに対して，StarFabricによるプロセッサ間通信の場合は，直接相手のローカル・メモリにデータを書き込むことができる（図19）．データの書き込みはPCIバス経由で行うため，転送速度に大きな差は生じない．しかし，CPUが書き込まれたデータを取得する場合には，自分のローカル・メモリに対してのアクセスだけとなるため，パフォーマンスを上げることができる．

図20に，複数のPCが相互にデータ通信を行うシステムの一例を示す．従来は，PC間のインターフェースとして高速な光インターフェースを使用していた．FibreChannelに代表される光インターフェースは非常に高価で，なおかつケーブルの引き回しなどにも注意が必要で，容易に使用することができない．

このシステムにStarFabricを適用した場合の構成を図21に示す．実効的なデータ転送速度には大差ないものの，スイッチ・デバイスを使用することで機材追加が容易になり，今後の拡張性が大きく広がる．また，StarFabricは光インターフェースと比較すると1/10程度の価格で使用することができるため，コストの面においても大きなアドバンテージをもっている．

● StarFabric対応デバイスの状況

現在，StarFabric対応デバイスとして，StarGen社から次の2製品がすでにリリースされている．

- SG1010： 6 Port StarFabric Switch

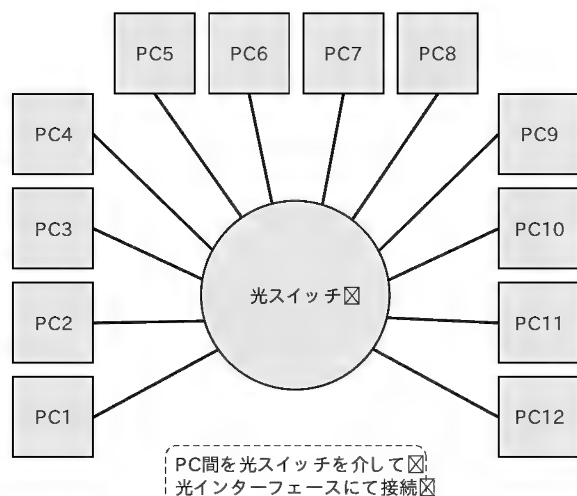


図20　光インターフェースによるPC間通信

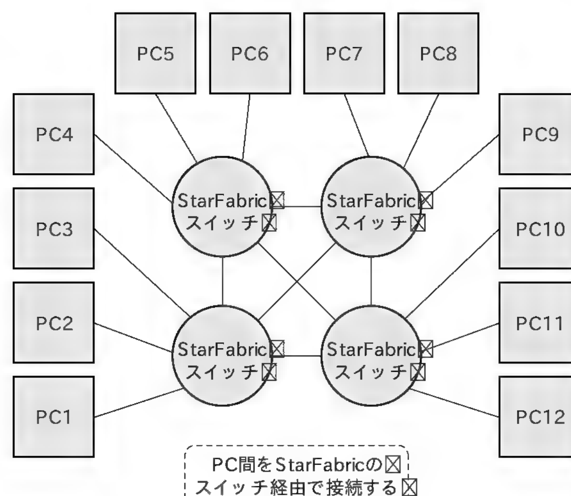


図21　StarFabricによるPC間通信

- SG2010：PCI to StarFabric Bridge

また，同社から評価用キットとして次の製品もリリースされている．

- SG1010 StarFabric Switch Development Kit
  ポート数：6ポート，16ポート
- SG2010 PCI to StarFabric Development Kit
  対応プラットホーム：PCI，CompactPCI，PICMG

このほかにも，StarFabric Trade Associationに参加している各社ボード・ベンダからも，対応製品がリリースされている．

● StarFabric関連の規格

StarFabric関連規格として，PCI Industrial Computer Manufacturers Group（PICMG）にて次の規格が策定されている．

- PCIMG 2.17：CompactPCI StarFabric Specification

また，VMEbus International Trade Association（VITA）では，VMEプラットホームへのStarFabricの実装方法に関する規格を現在策定中である．

- VITA 31.2：StarFabric on VME64x

おわりに

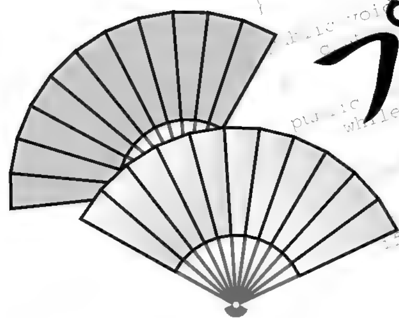
PCIバスに変わる高速I/OとしてPCI-XやPCI Expressがあるが，産業機器においてPCIバス以上の帯域を要求される場合は非常に少ない．

StarFabricはPCIバスの性能をカバーしているだけでなく，PCIバスにない機能を多く備えているため，システム性能を飛躍的に向上させることができる．また，PCIバスとの互換性を考慮してあり，既存システムとの親和性が非常に高く，今日まで積み上げてきたPCIバスの資産をそのまま利用できる．

さらに，デバイスが安価であり，実装もしやすいといった利点もある．

PCIバスの置き換えではなく，PCIバスを進化させる技術として，StarFabricを検討してみてはどうだろうか．

しらい・のの　（株）ロッキー





プログラミングの要

宮坂 電人

第12回

集合とハッシュ

集合(Sets)

配列や連結リストは，データをそのまま格納するしくみです．しかしデータが格納されるとき，そのデータがすでに配列や連結リストに格納されているかどうかの確認を怠ったために，二重に登録されてしまうことがあります．実際のプログラミングでは同じ値のデータが二つ以上格納されないよう要求されることがあります．たとえば住所録を考えてみましょう．山田太郎さんの情報は一つだけあれば十分であり，二つ以上あっても困ります．どちらの山田太郎さんの情報を信用していいのか困るからです．同じ値が二つ以上記録されないデータ構造を「集合」と称します．

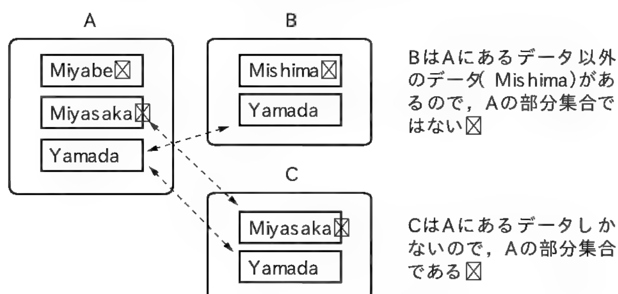
集合は初歩的なデータベース機能を要求されます．すなわち，

- 検索：すでにデータが登録済みであるか
- 登録：まだ登録されていないデータを登録する
- 削除：登録済みのデータを削除する

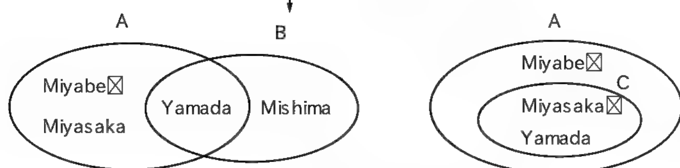
といったあたりです．また，集合どうしの照合(同じデータをかかえた集合どうしか，部分集合なのかなど)や合併(図1(a)，図1(b))も要求されるでしょう．具体的には以下のようなサービスが要求されるでしょう．

- `void insert(Object iObj)`：オブジェクトを登録する
- `Object remove(Object iObj)`：`iObj`と同値のオブジェクトがあれば，それを登録から外す．成功すれば登録されていたオブジェクトを返し，失敗すれば`null`を返す
- `boolean isMember(Object iObj)`：`iObj`と同値のオブジェクトがあれば`true`を返す
- `boolean isSubset(Sets iSets)`：自分が`iSets`の部分集合(subset)であれば`true`を返す
- `boolean isEqual(Sets iSets)`：自分が`iSets`と同値の集合であれば`true`を返す
- `int size()`：オブジェクトの登録数を返す
- `static Sets union(Sets iSet1, Sets iSet2)`：`iSet1`と`iSet2`の和集合(union)を作成して返す
- `static Sets intersection(Sets iSet1, Sets iSet2)`：`iSet1`と`iSet2`の積集合(intersection)を作成して返す
- `static Sets difference(Sets iSet1, Sets iSet2)`：

isSubset：部分集合を判定するメソッド

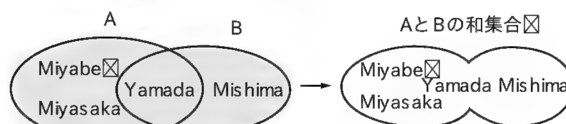


ベン図を使って表現すると

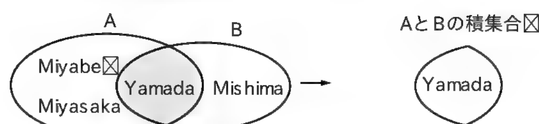


(a) ベン図による表現

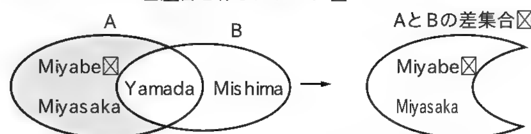
union：和集合を作るメソッド



intersection：積集合を作るメソッド



difference：差集合を作るメソッド



(b) 適用するメソッドによる違い

図1 集合のサービス

iSet1とiSet2の差集合(difference)を作成して返す．iSet1に登録されているがiSet2に登録されていないオブジェクトを新規集合とする

集合を実装するときは，登録しようとするデータの「値」を求め，それと同値のデータがすでに登録されていないかを検索するしくみが必要です．データを登録する機構自体は配列でも連結リストでもよいのですが，問題は検索の速度です．「Mastering Algorithms with C」注1では集合の実装に単方向連結リストを使った例を示しています．しかしながら，集合を実装するのに連結リストを使うのはあまり得策とはいえません．というのも，大量のデータを検索したり照合するのに連結リストはあまり向いていないからです．一応，筆者は本講座用に連結リストを使った集合の実装を作成しましたが，わざわざリストを見せる意味はないと判断しました．リストは今後の本誌に付属するCD-ROMにのみ収録します．

連結リストによる実装の欠点

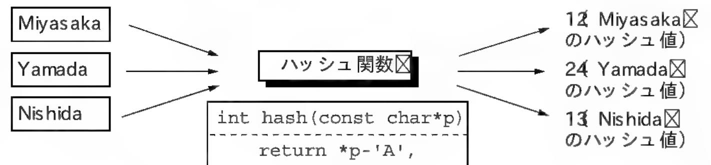
たくさんのデータを格納したいとき，通常は配列を使います．配列はシンプルな構造でわかりやすいのですが，ときにはそのシンプルさが足を引っぱって実行効率を落としたり，どのようにプログラミングすべきかで頭を悩ますことがあります．そのため，連結リストのような各要素をポインタで連結した構造がよく利用されます．配列と違って連結リストはサイズを拡大/縮小するのが得意であったり，途中への挿入や削除がすばやく行えるというメリットがあります．

ところが大量に記録すると検索に時間がかかるという欠点は，配列と同じです．むしろポインタをたどるステップがあることで，かえって遅くなる可能性もあります．実際のところ，検索はやっかいな問題です．目的のデータを探すために全データを巡回する必要があるからです．そのため記録しているデータが多ければ多いほど巡回処理に時間がかかるため，膨大な件数を検索するのは，どのようなデータ構造を取っても限界があるように思えます．

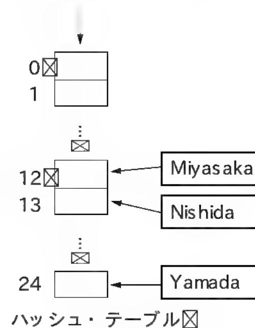
ハッシュ・テーブル

しかし，巡回処理を劇的に減らすことができれば，検索速度を劇的に減らすことができます．要するに全部を巡回するのではなく，一部だけを巡回するようにすれば，その分だけ速くなるのです．はたして，そんなつごうのよいことができるのかと疑う人もいるでしょう．

古くから使われている手法の一つで「ハッシュ・テーブル」があります．ハッシュ・テーブルの正体は配列です．配列のどの位置にデータを記録するかを決めるのは「ハッシュ関数」という特別な関数を使い，データの「ハッシュ値」を求め，その値に対応するインデックスにデータを登録します(図2)．登録されたデータを検索するときは，データのハッシュ値を求めるだけで登録位置が瞬時にわかります．つまり，登録された全部のデータをいちいち巡回しなくとも，ハッシュ値を求める1回の操作だけでデータの登録位置がわかるので高速に検索できるのです．



(注：このハッシュ関数はあくまで例であって実際にこのような単純な計算を行っている保証はない)



登録するデータのハッシュ値を求め，それを配列のインデックスとする

図2 ハッシュ・テーブルの原理

さて，ここでハッシュ関数とかハッシュ値という何やら難しげな用語がでてきましたが，原理はさほど難しくありません．電話帳の例で説明しましょう．仮に「山田太郎」の電話番号を探すとき，電話帳の最初のページから順番に探していくようなヒマ人はいません．普通は「や行」をめくって，そこから山田太郎さんを探すでしょう．ところが配列や連結リストで電話帳を作ってしまうと，山田太郎さんを検索するときに，いちいち「あ行」からたどってしまうという効率の悪い実装になってしまいます．この説明で出てきた「や行」というのがハッシュ値に相当するものです．そして，目的の人物名がどの行であるかを判断する処理が，ハッシュ関数に相当するものです．

実際にハッシュ関数をどう実装するかは，データの特性を考慮して決めなければなりません．理想的なハッシュ関数は，まんべんなくハッシュ値がばらつき，違うデータには違うハッシュ値が割り付けられ，ハッシュ値の衝突が起きないものですが，現実には，ほとんどありえない話です．電話帳の例でいえば，宮部みゆきのハッシュ値は「ま行」で，山田さんのハッシュ値と一致しないので問題ありませんが，宮坂電人のハッシュ値と衝突します．ハッシュ・テーブルの同じ位置に宮部みゆきと宮坂電人を登録できないので，どう処理するかという問題があります．このときには，2種類の選択肢があります．

● **連鎖法(Chained Hash Tables)**

衝突した位置に連結リストを格納する手法です(図3)．ハッシュ値を求める1回の操作だけでは済まず，そこからさらに連

注1：http://www.oreilly.com/catalog/masteralgoc/を参照．

結リストを巡回するので実行効率は落ちますが，それでも全部のデータを巡回するわけではないので高速に検索できるハッシュ・テーブルのメリットは生きてくるはずです．

● 開番地法（Open-Addressed Hash Tables）

衝突した位置から，わざとずれた位置の空き地を探し，そこにデータを格納する手法です．位置のずらしかたにはいくつか方法があるようですが，「Mastering Algorithms with C」では1個ずつ位置をずらして空き地を探す手法（図4）と，最初に使ったハッシュ関数とは別のハッシュ値を返す第2のハッシュ関数を用意し，両方の関数の戻り値を組み合わせる手法が紹介されています．

前者のずらしかたは，

インデックス値 = hash(p) + i

で，iを1からどんどん増やして空き地が見つかるまで繰り返す方法で，後者の方法は，

インデックス値 = hash(p) + i * hash2(p)

で，iを1からどんどん増やして空き地が見つかるまで繰り返すという違いがあります[注2]．

要求されるサービス

さて，ここでハッシュ・テーブルに要求されるサービスを考えましょう．さきほどの集合と同様，データの検索/登録/削除の機能を要求されるでしょう．具体的には以下のようなサービスが要求されるでしょう．

- `Object lookup(Object iObj)`：iObjと同値のオブジェクトを検索する
- `void insert(Object iObj)`：ハッシュ・テーブルにオブジェクトを登録する
- `Object remove(Object iObj)`：iObjと同値のオブジェクトが登録されているなら，それを削除する
- `int size()`：オブジェクトの登録数を返す
- `Object[] toArray()`：ハッシュ・テーブルに登録されているオブジェクトの配列を返す

toArrayメソッドは，「Mastering Algorithms with C」には掲載されていませんが，イテレータ的な処理を行いたいときや，全部の登録オブジェクトを簡単に確認したいときに重宝すると考えて実装したものです．

「Mastering Algorithms with C」ではコールバック関数として，

- `int (*h)(const void *key)`：ハッシュ関数．keyで示すデータのハッシュ値を返す
- `int(*match)(const void *key1, const void *key2)`：同値判定関数．key1とkey2が同値であるかどうかを返す

以上を初期化処理で指定させますが，Javaで実装する場合，Objectクラスに，

- `public int hashCode()`：ハッシュ値を求めるメソッド
- `public boolean equals(Object obj)`：objと同値であるかを判断するメソッド

があり，それぞれのクラスで必要に応じて，これらのメソッドをオーバライドすることになっています．この前提条件を，そのまま流用することにします．やっかいなのは開番地法を実装するときに第2のハッシュ関数を要求されるので，どうJavaで実装するかです．苦肉の策ですが，開番地法のハッシュ・テーブルに登録するオブジェクトは，リスト1のようなインターフェースを利用する前提にします．ちなみに筆者はテスト・プログラムを書くときIntegerクラスをよく利用するのですが，開番地法の確認で使ったInteger2なるクラスをリスト2のように実装しました．

連鎖法の実装

連鎖法ではハッシュ・テーブルの配列が連結リストの配列と

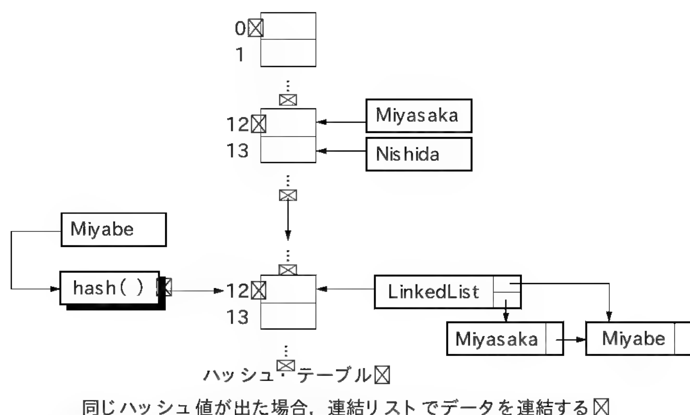


図3　連鎖法の原理

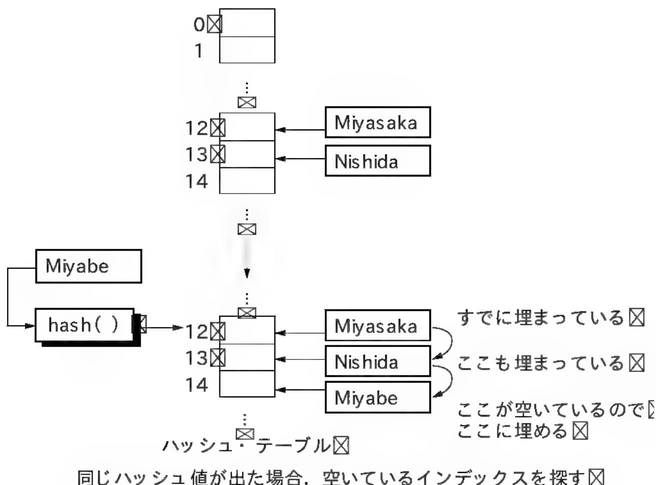


図4　開番地法の原理

注2：前者の方法は第2ハッシュ関数がつねに1を返すだけになっている後者の変形であるとも考えられる．

して実装されます．最低必要になるのは配列の要素数(mBuckets)と配列そのもの(mTable)です．また初期化処理では，あらかじめ配列の要素数を指定しておきます[注3]．

フィールドとコンストラクタはリスト3のように実装されます．

toArrayはハッシュ・テーブルの登録内容を配列にして返すメソッドです(リスト4)．mTableを巡回し，そこに連結リストがあるなら，さらに連結リストを巡回することで全登録内容が取得できます．

lookupはハッシュ・テーブルに同値のオブジェクトがあるかを探すメソッドです(リスト5)．目的のオブジェクトが配列(mTable)のどの位置にあるかをハッシュ値から求め(getBucketPosメソッドを使って)，そこから連結リストを巡回して探します．

リスト1 HashCode2.java

```
public interface HashCode2 {
    public int hashCode2(); //第2ハッシュ・コード取得メソッド
}
```

リスト2 Integer2.java

```
public class Integer2 implements HashCode2 {
    private Integer mInteger;

    //コンストラクタ
    public Integer2(int i){
        mInteger = new Integer(i);
    }
    //同値ならtrueを返す
    public boolean equals(Object iObj){
        if(iObj instanceof Integer2){
            Integer2 aObj = (Integer2)iObj;
            return aObj.intValue() == intValue();
        }
        return false;
    }
    //保持している値を返す
    public int intValue(){
        return mInteger.intValue();
    }
    //保持している値を文字列にして返す
    public String toString(){
        return mInteger.toString();
    }
    //第1ハッシュ・コード取得メソッド
    public int hashCode(){
        return intValue();
    }
    //第2ハッシュ・コード取得メソッド
    public int hashCode2(){
        return 7 + intValue() % 38711;
    }
}
```

リスト4 CHash.java(toArrayメソッド)

```
public Object[] toArray(){
    Object[] aArray = new Object[mSize];
    int aI = 0;
    for(int aBucket = 0; aBucket < mBuckets; aBucket++){
        Slist aList = mTable[aBucket];
        if(aList != null){
            for(SlistElmt aElmt = aList.getHead();
                aElmt != null; aElmt = aElmt.getNext()){
                aArray[aI++] = aElmt.getData();
            }
        }
    }
    return aArray;
}
```

insertはハッシュ・テーブルにオブジェクトを登録するメソッドです(リスト6)．最初にlookupメソッドで同値のオブジェクトがないかを調べ，ない場合には配列(mTable)のどの位置に登録するかを決めます．このとき，すでに連結リストがあるなら，そこに登録するだけですが，ない場合には先に連結リストを発生させます．

removeはハッシュ・テーブルから同値のオブジェクトを削除するメソッドです(リスト7)．削除できたのなら，そのオブジェクトの参照を返し，削除できなかったのならnullを返します．中の処理はlookupメソッドと似通っています．

sizeはオブジェクトの登録数を返すメソッドです(リスト8)．これはわざわざ説明するほどでもないでしょう．

開番地法の実装

連鎖法と違い，ハッシュ・テーブルの配列にはオブジェクトそのものが登録されます．ちょっとやっかいなのは，登録した

リスト3 CHash.java(フィールドとコンストラクタ)

```
public class CHash {

    private int mBuckets;   //バケツ数
    private Slist[] mTable; //連結リストの配列
    private int mSize;      //オブジェクトの登録数

    //コンストラクタ
    public CHash(){
        initCHash(0x1000);
    }
    public CHash(int iBuckets){
        initCHash(iBuckets);
    }
    private void initCHash(int iBuckets){
        mBuckets = iBuckets;
        mTable = new Slist[iBuckets];
        mSize = 0;
    }
```

リスト5 lookupメソッド

```
public Object lookup(Object iObj){
    //バケツ位置を求める
    int aBucket = getBucketPos(iObj);
    Slist aList = mTable[aBucket];
    if(aList != null){
        //バケツ位置の連結リストからオブジェクトを探す
        for(SlistElmt aElmt = aList.getHead();
            aElmt != null; aElmt = aElmt.getNext()){
            Object aObj = aElmt.getData();
            if(aObj.equals(iObj)){
                return aObj;
            }
        }
    }
    return null;
}
private int getBucketPos(Object iObj){
    int aHashCode = iObj.hashCode(); //ハッシュ値を求める
    return ((aHashCode < 0) ? -aHashCode : aHashCode)
            % mBuckets;
}
```

注3：実際に利用されるハッシュ・テーブルでは，あらかじめデフォルト値が決まっていて外部から指定できないようにしていたり，登録数が増えたなら自動的に配列を拡大する実装もある．

リスト 6　insertメソッド

```
public void insert(Object iObj){
    //すでに登録ずみかnullなら戻る
    if(iObj == null || lookup(iObj) != null){
        return;
    }
    //バケツ位置を求める
    int aBucket = getBucketPos(iObj);
    //バケツ位置に連結リストが用意されていないなら
    // 連結リストを発生させる
    Slist aList = mTable[aBucket];
    if(aList == null){
        aList = new Slist();
        mTable[aBucket] = aList;
    }
    //バケツ位置の連結リストにオブジェクトを登録する
    aList.insertNext(null,iObj);
    ++mSize;
}
```

リスト 8　sizeメソッド

```
public int size(){
    return mSize;
}
```

リスト 10　OHash.java(toArrayメソッド)

```
public Object[] toArray(){
    Object[] aArray = new Object[mSize];
    int aI = 0;
    for(int aBucket = 0; aBucket < mBuckets; aBucket++){
        Object aObj = mTable[aBucket];
        if(aObj != null && aObj != mVacated){
            aArray[aI++] = aObj;
        }
    }
    return aArray;
}
```

リスト 11　OHash.java(lookupメソッド)

```
public Object lookup(Object iObj){
    for(int aI = 0; aI < mBuckets; aI++){
        //バケツ位置を求める
        int aBucket = getBucketPos(iObj,aI);
        //そこの位置のオブジェクトをえる
        Object aObj = mTable[aBucket];
        //nullなら，これ以上探しても見つからない
        if(aObj == null){
            return null;
        }
        //無効オブジェクトでなく同値であるなら，
        // そのオブジェクトが答
        if(aObj != mVacated && aObj.equals(iObj)){
            return aObj;
        }
    }
    return null;
}
private int getBucketPos(Object iObj,int iPos){
    int aHashCode = iObj.hashCode(); //ハッシュ値を求める
    if(iPos != 0){
        int aHashCode2 = ((HashCode2)iObj).hashCode2();
        aHashCode += (iPos * aHashCode2);
    }
    aHashCode = ((aHashCode < 0) ? -aHashCode : aHashCode)
                % mBuckets;
    return aHashCode;
}
```

オブジェクトを削除するときです．というのも，削除した場所にnullを埋めると，検索の打ち切り場所だと誤解されるおそれがあるからです．したがって，削除した場所にはnullではなく無効の目印となるオブジェクト(mVacated)を埋めておきます．

フィールドとコンストラクタはリスト 9のように実装されます．

リスト 7　removeメソッド

```
public Object remove(Object iObj){
    //バケツ位置を求める
    int aBucket = getBucketPos(iObj);
    Slist aList = mTable[aBucket];
    if(aList != null){
        SlistElmt aPrev = null;
        //バケツ位置の連結リストからオブジェクトを探す
        for(SlistElmt aElmt = aList.getHead();
            aElmt != null; aElmt = aElmt.getNext()){
            if(aElmt.getData().equals(iObj)){
                --mSize;
                return aList.removeNext(aPrev);
            }
            aPrev = aElmt;
        }
    }
    return null;
}
```

リスト 9　OHash.java(フィールドとコンストラクタ)

```
public class OHash {
    //無効オブジェクトの印
    private static Object mVacated = new Object();

    private int mBuckets;         //バケツ数
    private Object[] mTable;      //登録オブジェクトの配列
    private int mSize;            //オブジェクトの登録数

    //コンストラクタ
    public OHash(int iBuckets){
        mBuckets = iBuckets;
        mTable = new Object[iBuckets];
        mSize = 0;
    }
```

toArrayメソッドでmTableを巡回するのは連鎖法と同様ですが，連結リストではなくオブジェクトそのものが登録されているので実装はシンプルになります．しかし，無効オブジェクト(mVacatedまたはnull)を避ける処理が必要になります(リスト 10)．ほかのメソッドについても同値のオブジェクトを探すロジックが凝っているため，連鎖法より若干ややこしくなります(リスト 11～リスト 13)．

ハッシュ・テーブルを利用した集合の実装

ハッシュ・テーブル(連鎖法)を使って集合を実装してみます．ハッシュ・テーブル自体が集合と似たような機能をもっているため，登録/検索/削除については内部にかかえたハッシュ・テーブル・オブジェクトのメソッドを呼ぶだけで実装できます(リスト 14)．

しかし，集合どうしの照合/合併となると若干手間がかかります．ハッシュ・テーブル内の巡回が必要になるからです．連鎖法で実装した場合，巡回そのものが凝ったロジックになります．しかしtoArrayメソッドを利用したり，前回紹介したiteratorメソッドを実装することで，ややこしい巡回処理から逃れられるはずです．ここではせっかくtoArrayメソッドを作ったのですから，このメソッドを利用しています(リスト 15)．

みやさか・でんと　miyadent@anet.ne.jp

リスト 12　OHash.java（insert メソッド）

```
public boolean insert(Object iObj){
    //nullか，登録容量を越えているか，hashCode2メソッドが
    // 利用できないなら戻る
    if(iObj == null || mSize >= mBuckets
        || !(iObj instanceof HashCode2)){
        return false;
    }
    //すでに登録ずみなら戻る
    if(lookup(iObj) != null){
        return true;
    }
    for(int aI = 0; aI < mBuckets; aI++){
        //バケツ位置を求める
        int aBucket = getBucketPos(iObj,aI);
        //そこの位置に無効オブジェクトがあるなら入れ替える
        if(mTable[aBucket] == null || mTable[aBucket]
            == mVacated){
            mTable[aBucket] = iObj;
            ++mSize;
            return true;
        }
    }
    return false; //ここを通過することはないはずだが念のため
}
```

リスト 13　OHash.java（remove メソッド）

```
public Object remove(Object iObj){
    for(int aI = 0; aI < mBuckets; aI++){
        //バケツ位置を求める
        int aBucket = getBucketPos(iObj,aI);
        //そこの位置のオブジェクトをえる
        Object aObj = mTable[aBucket];
        //nullなら，これ以上探しても見つからない
        if(aObj == null){
            return null;
        }
        //無効オブジェクトでなく同値であるなら，
        // そのオブジェクトが削除対象
        if(aObj != mVacated && aObj.equals(iObj)){
            mTable[aBucket] = mVacated;
            --mSize;
            return aObj;
        }
    }
    return null;
}
```

リスト 14　Sets.java（フィールドとおもなメソッド）

```
public class Sets {
    private CHash mTable; //オブジェクトを格納する場所

    //コンストラクタ
    public Sets(){
        mTable = new CHash();
    }
    //登録オブジェクトを配列にして返す
    public Object[] toArray(){
        return mTable.toArray();
    }
    //オブジェクトを登録する
    public void insert(Object iObj){
        mTable.insert(iObj);
    }
    //iObjと同値のオブジェクトがあればtrueを返す
    public boolean isMember(Object iObj){
        return (mTable.lookup(iObj) != null);
    }
    //iObjと同値のオブジェクトがあれば，それを登録から外す
    //成功すれば登録されていたオブジェクトを返す
    // 失敗すればnullを返す
    public Object remove(Object iObj){
        return mTable.remove(iObj);
    }
    //オブジェクトの登録数を返す
    public int size(){
        return mTable.size();
    }
```

リスト 15　Sets.java（集合独自のメソッド）

```
//自分がiSetsの部分集合(subset)であればtrueを返す
public boolean isSubset(Sets iSets){
    //自分の登録数がiSetsの登録数より多いなら部分集合ではない
    if(size() > iSets.size()){
        return false;
    }
    //自分の登録オブジェクトがiSetsに登録されているかを
    // 確認する
    Object[] aObjs = mTable.toArray();
    for(int aI = 0; aI < aObjs.length; aI++){
        if(!iSets.isMember(aObjs[aI])){
            //登録されていないなら部分集合ではない
            return false;
        }
    }
    return true; //部分集合である
}
//自分がiSetsと同値の集合であればtrueを返す
public boolean isEqual(Sets iSets){
    //自分の登録数がiSetsの登録数と同じでないなら
    // 同値の集合ではない
    if(size() != iSets.size()){
        return false;
    }
    //自分がiSetsの部分集合であるなら同値の集合となる
    return isSubset(iSets);
}
//iSet1とiSet2の和集合(union)を作成して返す
public static Sets union(Sets iSet1,Sets iSet2){
    Sets aSetU = new Sets();
    Object[] aObjs;
    int aI;
    //iSet1の登録オブジェクトを新規集合に登録する
    aObjs = iSet1.toArray();
    for(aI = 0; aI < aObjs.length; aI++){
        aSetU.insert(aObjs[aI]);
    }
    //iSet2の登録オブジェクトを新規集合に登録する
    aObjs = iSet2.toArray();
    for(aI = 0; aI < aObjs.length; aI++){
        aSetU.insert(aObjs[aI]);
    }
    return aSetU;
}
//iSet1とiSet2の積集合(intersection)を作成して返す
public static Sets intersection(Sets iSet1,Sets iSet2){
    Sets aSetI = new Sets();
    //iSet1の登録オブジェクトがiSet2にも登録されているなら，
    // それを新規集合に登録する
    Object[] aObjs = iSet1.toArray();
    for(int aI = 0; aI < aObjs.length; aI++){
        Object aElmt = aObjs[aI];
        if(iSet2.isMember(aElmt)){
            aSetI.insert(aElmt);
        }
    }
    return aSetI;
}
//iSet1とiSet2の差集合(difference)を作成して返す
//iSet1に登録されているがiSet2に登録されていない
// オブジェクトを新規集合とする
public static Sets difference(Sets iSet1,Sets iSet2){
    Sets aSetD = new Sets();
    Object[] aObjs = iSet1.toArray();
    for(int aI = 0; aI < aObjs.length; aI++){
        Object aElmt = aObjs[aI];
        if(!iSet2.isMember(aElmt)){
            aSetD.insert(aElmt);
        }
    }
    return aSetD;
}
```

組み込みプログラミング・ノウハウ入門（第15回）

$f(a+b) \leqq f(a)+f(b)$の秘密——反復開発とアーキテクチャ

藤倉 俊幸

繰り返し開発，あるいは反復開発ということばは，今では広く認識されるようになった．しかし，実際に行った経験のある人はそれほど多くはない．経験してうまくいった人はさらに少ないのではなかろうか．

うまくいかない原因はいろいろある．よくある事例としては，繰り返すたびに以前に作った部分の手直しが必要になり，この作業がオーバヘッドとなり，だんだん開発効率が落ちていく．そして最後には，繰り返す時間的余裕がなくなってしまうというものだ．たとえば1回の繰り返しを3週間として，3回繰り返すとする．1回目は予定どおりでも，2回目の繰り返しは5週間になり，3回目が6週間になってしまう．するとトータルで5週間の遅れになる．これが，よくある事例であり，本に書いてあるようにはいかないのが現実だ．今回はこの問題について考えてみる．

繰り返しごとにズレが拡大するパターンにはいろいろあるが，図1のように，実装Iとテスト Tが延びていく場合は，前の繰り返しで作った部分の作り変えが発生している場合が多い．これが，繰り返しのオーバヘッドである．

一方，繰り返すごとに設計の部分Dが延びていく，または短くならない場合は，アーキテクチャに問題があることが多い．アーキテクチャが安定していない場合には，繰り返し開発は向かない．設計部分Dがだんだん長くなるということは，それはすなわち，繰り返しごとにアーキテクチャを作り直していることになる．この場合には，ほとんどの部分が作り変えになるので，実装Iとテスト Tもさらに延びてしまう．

RUP（Rational Unified Process）の場合，繰り返しをフェーズで分けているが，アーキテクチャを安定化する推敲フェーズから詳細の作りこみに入る作成フェーズへの移行が早すぎた可能性がある．要求定義Rが短くならない場合には，開発に入ることそのものが早すぎた可能性がある．どこが延びるかによって原因が異なるので，開発のメトリクスを取ることは重要である．

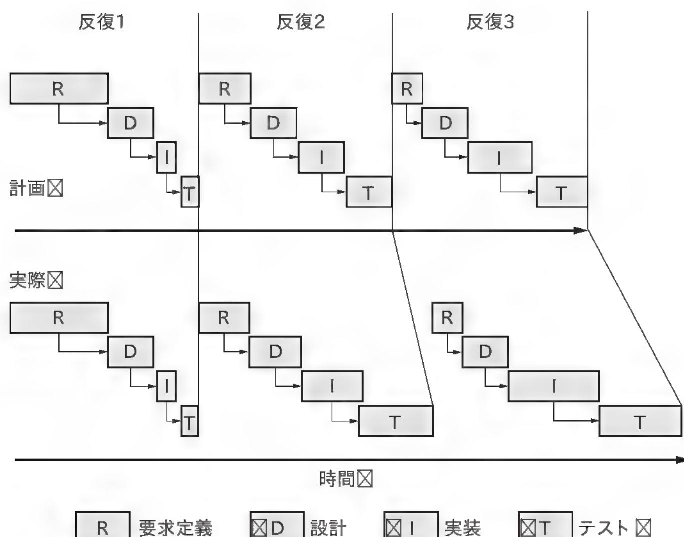


図1 反復開発の計画と実際のズレ

1 要求の分解と実装の構築

RUPの場合，要求はUse Case単位に分解されて管理される．ソフトウェアを作っていく単位はこのUse Caseになる．これから開発すべきソフトウェアが満たすべき要求全体をReqで表し，Reqを分解した各Use CaseをUC_1，UC_2，…，UC_nで表すと，

$$Req = UC_1 + UC_2 + UC_3 + \cdots + UC_n \quad \cdots\cdots\cdots\cdots\cdots (1)$$

と表現できる．このように分解された要求Reqを繰り返し開発で開発するということは，開発行為をDepという関数で表現すると，以下のようになる．

繰り返し1回目成果物 $\leftarrow Dep_1(UC_1)$
繰り返し2回目成果物 $\leftarrow Dep_1(UC_1) + Dep_2(UC_2)$
繰り返し3回目成果物 $\leftarrow Dep_1(UC_1) + Dep_2(UC_2) + Dep_3(UC_3)$
⋮
最終成果物 $\leftarrow Dep_1(UC_1) + Dep_2(UC_2) + Dep_3(UC_3) + \cdots + Dep_n(UC_n)$

たとえば，$Dep(\ )$が開発の工数を表す関数であると解釈して，

$$Dep(Req) = Dep_1(UC_1) + Dep_2(UC_2) + Dep_3(UC_3) + \cdots + Dep_n(UC_n) \quad \cdots\cdots\cdots\cdots\cdots (2)$$

となれば一括で開発しても，繰り返し開発で少しずつ開発して

もオーバヘッドはないことになる．しかし，実際はそう簡単ではない．つまり，

$$Dep(Req) \leqq Dep(UC_1) + Dep(UC_2) + Dep(UC_3) + \cdots + Dep(UC_n) \quad \cdots\cdots (3)$$

となる．式(1)の両辺に $Dep(\ )$ を施すとすれば，つまり繰り返し開発ではなく，まとめて作るとすれば，

$$Dep(Req) = Dep(UC_1 + UC_2 + UC_3 + \cdots + UC_n) \quad \cdots\cdots (4)$$

となる．しかし式(3)から，

$$Dep(UC_1 + UC_2 + \cdots + UC_n) \leqq Dep(UC_1) + Dep(UC_2) + \cdots + Dep(UC_n) \quad \cdots\cdots (5)$$

ということになる．簡単のために UC_1 と UC_2 だけを考えると．

$$Dep(UC_1 + UC_2) \leqq Dep(UC_1) + Dep(UC_2)$$

となるが，これはたとえば，

$Dep(UC_1 + UC_2) =$ 100H　/* まとめて開発した場合 */
$Dep(UC_1) =$ 50H　/* UC_1 のみ開発した場合 */
$Dep(UC_2) =$ 40H　/* UC_2 のみ開発した場合 */
$Dep(UC_1) + Dep(UC_2) =$ 120H
/* UC_1 を作ってから，UC_2 を作った場合 */

のような感じになる．$Dep(UC_2)$ の作業の中に $Dep(UC_1)$ で作った部分の見直しや作り直しが含まれるためである．$Dep(UC_1 + UC_2)$ の場合は，いっしょに作るので UC_1 だけで動くソフトウェアを作りテストする必要がないので短くなる．ついでにいうと，開発順序によっても開発工数は変わるだろう．たとえば，

$Dep(UC_2) + Dep(UC_1) =$ 130H
/* UC_2 を作ってから，UC_1 を作った場合 */

では繰り返し開発しないほうが良いと思うかもしれないが，Use Caseの数が多くなってくると，まとめて作ることは，複雑すぎて不可能になるとか，設計が収束しなくなってしまうといわれている．それから，開発がスタートしてから判明した仕様の不足や変更を，次の繰り返しで吸収することができる．これらのことから，繰り返し開発は効果があると一般に認識されている．ただし，その効果がオーバヘッドに負けてしまうことが比較的多いのである．オーバヘッドというよりは過大な期待に負けてしまうといったほうが正確かもしれない．

● XPの場合は

RUPの対極にあるXP(Extreme Programing)では，オーバヘッドにより失敗した話はあまり聞かない．もっとも，事例自身が少ないのかもしれない．XPではUse Caseの代わりに要求をインデックス・カードやストーリ・カードなどに分解して，繰り返し開発をする．この場合でもオーバヘッドがあることは同様である．

ただし，ペア・プログラミングやコードの共同所有などのように，「チームでまとめて作る」的なところがあるため，じつは式(4)の世界から踏み出していないのかもしれない．XPではアーキテクチャなどの重要な用件は，プログラマ間のコミュニケーションの中で自然発生的にできあがることを期待している．XPがうまくいっているときは，初期の反復でアーキテクチャが定まってくるので，後半の反復の効率が向上する．また，リファクタリングによってもアーキテクチャの見直しが行われる．その際に，全体をリードするアーキテクト的なプログラマが出現するが，アーキテクチャ自身は全員が共同で作り共有する形をとる．したがって，繰り返すといっても"みんなでまとめて作る"的なところがある．

XPの要求分割の粒度はUse Caseより小さく，Use Caseの個々のシナリオに対応すると考えられる．というか繰り返し単位の開発期間があらかじめ決めた反復周期に収まらないようであれば，合うようにさらにタスクに分割する．とにかくUse Caseよりは小さい．要求間の関係や実装順序などはオンサイト・カスタマという一種の人質(？)の頭の中にある．オンサイト・カスタマが，しゃべる要求仕様書であり，しゃべるUse Caseである．つまり，これが一人の頭の容量が開発できるソフトウェア・サイズの上限になる．一般的に，RUPで問題になる不可避なオーバヘッドが顕在化する前に，オンサイト・カスタマが飽和するのかもしれない．

2 Schwarzの不等式とミクロ経済学の劣加法性

式(5)の形の不等式はどこかで見たことがある．シュワルツ[注]の不等式である．

この不等式は別名，三角不等式ともいう．任意の3点 x，y，z について次の不等式が成り立つ．ここで ρ は2点間の距離である(図2)．

$$\rho(x, z) \leqq \rho(x, y) + \rho(y, z)$$

意味は明確で x から z に行くのに y を通っていくと距離が長くなるということである．シュワルツの不等式は重要な不等式で，以下のようにいろいろな形でいろいろなところに出てくる．

$$(ax - by)^2 \leq (a^2 + b^2)(x^2 + y^2) \qquad |(x, y)| \leq \|x\|\|y\| \quad \cdots\cdots (6)$$

ソフト開発の費用という意味では，ミクロ経済学の劣加法性

注：余談ではあるが，確認しようと思って岩波の理化学辞典で調べたら，シュヴァルツの不等式になっていた．ワとヴでは，ページがだいぶ違うので余計な時間を使ってしまった．ちなみに，岩波の数学辞典の方にはカタカナ読みは載ってなかった．外国人の名前表記は何通りもある．「ジキル博士とハイド氏」や「宝島」を書いたスティーヴンソンの表記が17通りもあると調べた人がいる[1)]．また全国の図書館の図書カードで外国人の名前がどのように表記されているかまとめた辞典もある．電子辞書を引いたり，Webで検索したりすることを考えるとカタカナ表記だけではヒットしないことがある．曖昧(あいまい)検索技術が進んでも，検索に時間がかかるのでオリジナルな綴りを併記しておいてもらいたい．ところで，英語以外のことばを英語で表記する際にはバラつきはないのだろうか．

がおもしろい．特定の企業がどんどん大きくなり自然独占の状態になるのは，費用の劣加法性が成立している場合である．別の言いかたをすると式(5)が成立していて，会社を大きくしたほうがコストが掛からない場合である．劣加法性が成立していないのに，特定の企業が大きくなるのは自由な競争が行われていない場合で，社会全体のコストが増えてしまう．これは不自然な独占状態である．余分なコストは税金を投入したり，ユーザが高い料金を払わされたりすることで埋め合わせされる．

たとえば，長距離通話と市内通話では，長距離通話において劣加法性が成立しないことが示されたので，NTTやAT&Tは地域会社に分割された．しかし，費用関数の推定手法を変えると逆の結論が得られることもあり，なかなか簡単にはいかないようである．電話と鉄道は過去の話になってしまったが，道路と郵便はこれからなので説明が付くようにしてもらいたい．

振り返ってソフト開発でも費用関数を推定し劣加法性が成り立つか否かで，本来は繰り返し開発を行うべきかどうかを決めるべきではないだろうか．自由競争を導入する部分と独占状態を維持する部分とで市場境界を定義するように，一つのプロジェクトの中でも繰り返す部分とそうでない部分を識別できるのではないだろうか．また，ユースケースの粒度なども最適な大きさが存在しているはずである．

● 等号が成り立つ場合

シュワルツの不等式で等号が成り立つのは，図2でいえば点yが線分xz上にあるときである．式(6)では，$x=a$，$y=b$とか$x=y$などが成立する場合である．これでは，ソフト開発に当てはめると，同じUse Caseを作る場合になってしまうので意味がない．

使えそうな構造は確率統計の分野にある．期待値の加法性や分散の加法性である．確率変数X，Yの和$X+Y$の期待値Eについて，

$$E(X+Y)=E(X)+E(Y)$$

が成り立つ．また，XとYが互いに独立であれば分散Vについても，

$$V(X+Y)=V(X)+V(Y)$$

が成り立つ．互いに独立な，相互作用のない部分同士であればオーバヘッドがないのは当然かもしれない．

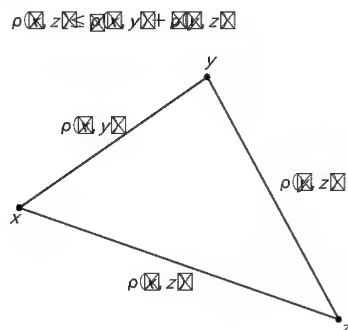


図2　三角不等式の意味

ソフトウェアを設計する場合は，なるべく依存関係が複雑にならないようにモジュール分割やパッケージ分割を行うことが基本である．独立な部分に分割できれば，繰り返し開発でも，複数人が一括で並行開発する場合でも同じになる．モジュール分割は，アーキテクチャの一部である．アーキテクチャが重要という意味は，独立な部分に分割するというレベルでは繰り返し開発と並行開発では差がない．

3 インターフェース，構造，ふるまい

ソフトウェアを構成する三要素は，インターフェースと構造とふるまいである(表1)．インターフェースは関数のシグネチャ，パブリック変数などである．構造はクラスや構造体などで，ふるまいは関数の中身などが該当する．ヘッダ・ファイルに記述されるのは，インターフェースのすべてとそのインターフェースから参照される構造である．ヘッダ・ファイルに書かれているこれらの内容が，ソフトウェア・アーキテクチャを構成すると思えばよい．つまり，

ヘッダ・ファイルの内容(インターフェース＋静的構造)
→アーキテクチャ

である．ただし，アーキテクチャにはタスク分割の仕方などの動的な構造などが含まれている．そして，これらは必ずしもヘッダ・ファイルには書かれないので逆は成立しない．構造の中に動的なものも含めて，

インターフェース＋構造＝アーキテクチャ

となる．動的な構造は，並列性や実行の順序や時間制約に関する部分でソース・コード内に分散してしまう．アクティブ・オブジェクトは，ふるまいと密に結合してしまう動的構造をカプセル化する手段になる．

● $f(a+b)\fallingdotseq f(a)+f(b)$となるもの

ユーザは，要求項目を挙げてくる．要求項目は，自然言語で書かれた単なるリストなので追加は簡単にできてしまう．そのため，

$$f(a+b)\fallingdotseq f(a)+f(b) \quad \cdots\cdots\cdots\cdots (7)$$

が成立する．実際には，要求を追加する際に既存の要求項目と矛盾がないかなどを検討しなければならないので，簡単に追加できるとはいえない．しかし，矛盾が生じた場合は，新しい場合分けの数が増えるだけなので，ユーザ・レベルでは追加は容易である．

要求のほかには，インターフェースと静的な構造でも式(7)

表1　ソフトウェア構成要素

インターフェース	構　造	ふるまい
外部関数宣言 外部変数宣言	クラス・構造体 プロセス・スレッド優先度 モジュール構成	関数の中身 ステート・マシン スレッド動作

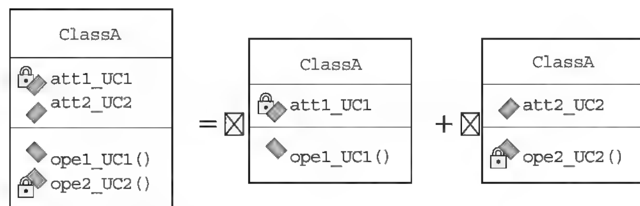


図3　クラス宣言のマージ

表2　反復に対するオーバヘッド

少ない：$f(a+b) \fallingdotseq f(a)+f(b)$	多い：$f(a+b) < f(a)+f(b)$
要求の追加 インターフェースの追加 静的構造の追加	ふるまいの追加 インターフェースの変更・分割 静的構造の分割 動的構造の追加・分割

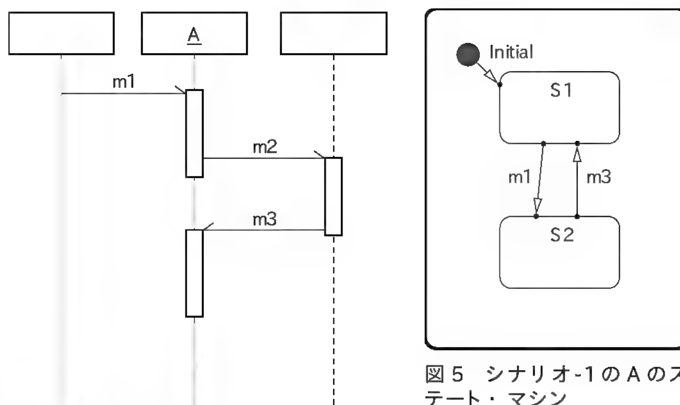


図4　シナリオ-1

図5　シナリオ-1のAのステート・マシン

が成立する．名前が衝突しない限り，ヘッダ・ファイルはマージしたり，インクルードできるので，このことは自明である．クラス宣言でも同様である．図3のようにマージできる．

要するにインターフェースと静的な構造は，繰り返し開発のオーバ・ヘッドにはほとんどならずに機械的に追加できるのである．

● $f(a+b) < f(a)+f(b)$となるもの

一方，ふるまいの追加・変更は機械的にはできない．すでに作った関数に対して新しい動作条件が追加された場合，どこかに`if`文を入れて制御を分岐させて，新しい条件下での処理を実行してまた制御を合流させなければならない．ふるまいに対しては式(7)ではなく，

$$f(a+b) < f(a)+f(b) \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots (8)$$

が成立する．複雑な処理を行う大きな関数を作るよりは，単機能な小さな関数でソフトウェアを構築しておけば，機能追加は関数の中身を変更せずに，別の小さな関数を作ることで対応できる確率が高くなる．別途新しい関数を作る場合は，機能をマージするわけではないのでオーバヘッドは少なくなる．関数の数を増やすことに抵抗があるために別途関数を作らず，既存の関数を変更した場合には，必然的にその関数は大きくなる．多機能な関数になればなるほど，的が大きくなると弾に当たりやすくなるのと同様で，機能追加のときに巻き込まれる可能性が高くなり，その結果また関数が大きく多機能になっていく．この悪循環に陥ると，オーバヘッドが繰り返しのたびに増えていく．つまり，次の繰り返しで頑張ってスケジュールを立て直そうと思っても，次ではさらにスケジュールは遅れるのである．

関数の中身が肥大化するだけであればまだ良いが，引き数が追加されるような変更を行うとインターフェースが変更されてしまう．既存のインターフェースが変更されるとアーキテクチャの変更になるので，ソフトウェア全体や開発チーム全体が影響を受けてしまう．繰り返し開発をする場合には，機能を追加することのオーバヘッドが全体に波及しないように，ポリモフィズムを利用するなどのリファクタリング手法[2)]を利用する必要がある．たとえば，インターフェースでは，引き数としてスーパ・クラスを指定しておくなどの方法である．タスク間で，メッセージの送受信を行う場合でもメッセージのデータ型が変更になると，インターフェースが変更になったのと同じで影響が大きい．しかも，多くのRTOSはC++への対応が遅れているのでスーパ・クラスを渡すなどの手法が使えない．したがって，インターフェースで使用するデータ構造を早めに確定することが重要である(表2)．

4 アクティブオブジェクトによるふるまい記述

繰り返し開発でオーバヘッドの原因になる主要な要因は，ふるまいの記述である．ふるまいの場合は，インターフェースや静的構造と違って，追加するだけでも作り変えのオーバヘッドをともなう．ふるまいでもデータを処理するアルゴリズムなどの末端のふるまいは，別の関数として追加するなどの比較的オーバヘッドの少ない機能追加が可能である．しかし，ソフトウェア全体を制御する動的構造を実装したふるまいの部分では，どうしても既存の関数の変更が必要になってくる．たとえば，RUPの場合にはシナリオ，XPではストーリを追加する際に既存のコントロール・クラスを変更する確率は非常に高い．

この場合のオーバヘッドを減らすために，前回と前々回の説明でも使用したプロセス代数を利用するステート・マシンの自動生成が役に立つ．特に追加されるシナリオが今までのシナリオと並列で動かなければならない場合には，ほとんどの場合，ステート・マシンを作り変えなければならない．たとえば，図4のシナリオのAのふるまいを実装したステート・マシンは図5のようになる．

次の反復の際に，図6のシナリオをAに対して追加しなけれ

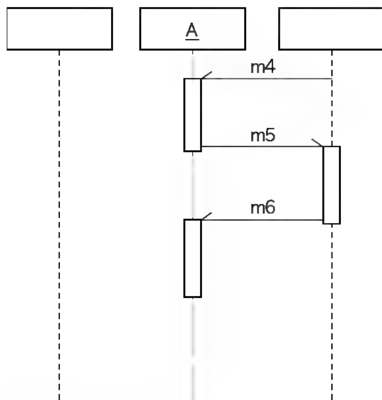


図6　シナリオ-2

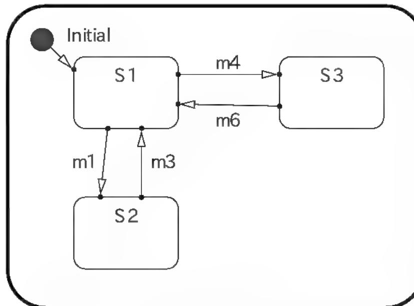


図7　シナリオ-1とシナリオ-2が排他的な場合

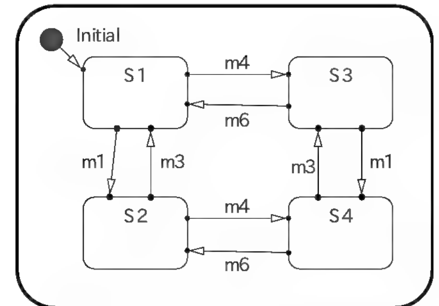


図8　シナリオ-1とシナリオ-2が並列な場合

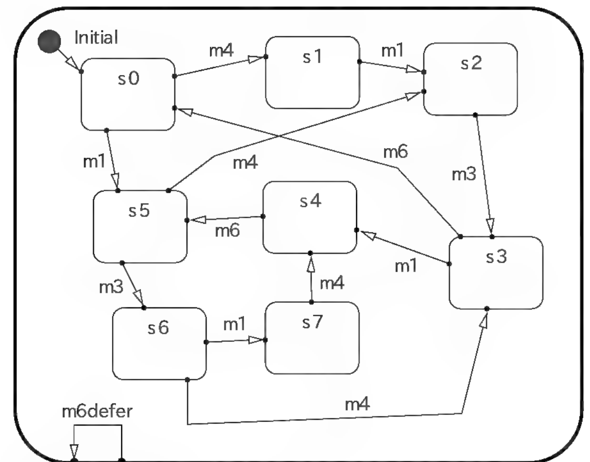


図9　m3->m6の順序制約がある場合

ばならない場合を考える．このとき，図5のステート・マシンは作り変えになってしまう．二つのシナリオが排他的であれば，比較的簡単に新しいステート・マシンを作ることができる．排他的という意味は，シナリオ-1が終了してからシナリオ-2が始まる場合である．この場合のステート・マシンは図7のようになる．また，排他的でない場合には図8のようになる．

このあたりまでは比較的簡単に式(7)のレベルで機能追加できるが，実際のシステムではシナリオが完全に排他的だったり並列だったりすることはまれである．たいていは部分的に処理の順番が指定されていることが多い．たとえば，

制約1：m3の処理が終了した状態でのみm6の処理を実行できる

という制約が付いていた場合，今までのステート・マシンの構造はまったく再利用できなくなる．その結果，式(8)のレベルのオーバヘッドが掛かってくる．実際のステート・マシンは，制約1がアクティブ・オブジェクトの内部的な制約なのか，外部的な制約なのかによって変わる．内部制約の場合のステート・マシンは図9のようになる．

図4と図6のような単純なシナリオでさえ並列性が絡むと繰り返しのオーバヘッドは非常に大きくなる．したがって，並列動作が当然の組み込みシステムでは，繰り返し開発は実施が困難になる場合が多い．最初に可能なシナリオを洗い出してから設計しなければ，オーバヘッドに耐えられなくなる．つまり，最初に要求項目を全部確定してから設計に入るウォータ・フォール開発になってしまうのである．

しかし，要求を確定することは実質的に不可能で，要求変更や追加は必ず発生する．これらの変更に対応できなければ開発が終了したときには，すでに時代遅れのシステムになっている可能性がある．また，開発中にハードウェアのバグが発見されてソフトウェアで回避しなければならない場合などもあり，ウォータ・フォール開発も実際には不可能である．

では，どうするか．シナリオ-1のアクティブ・オブジェクトAの動作は，前回紹介したLTSAツール[3]で表現すると，

$$A1 = (m1\text{->}m3\text{->}A1) \quad \cdots\cdots (9)$$

となる．またシナリオ-2(図6)の動作は，

$$A2 = (m4\text{->}m6\text{->}A2) \quad \cdots\cdots (10)$$

となる．この二つのシナリオを並列動作させるのであればAの動作は，並列化オペレータで結合したものになる．

$$\|A = (A1 \| A2) \quad \cdots\cdots (11)$$

ステート・マシンは式(11)から生成すれば図8を得ることができる．式(9)と式(10)を式(11)に代入し，制約1を式で表すと，

$$R1 = (m3\text{->}m6\text{->}R1) \quad \cdots\cdots (12)$$

となるので式(11)に式(12)の$R1$を追加してステート・マシンを生成すれば図9を得ることができる．具体的には，図10に示したようにLTSAツールにプロセス式をコンパイル/合成させてラベル遷移システムを生成させて，使用する制約の種類を考慮しながら生成されたラベル遷移システムを，実際に使用するステート・マシンのセマンテックスに変換する．

$$\|A = (A1 \| A2 \| R1) \quad \cdots\cdots (13)$$

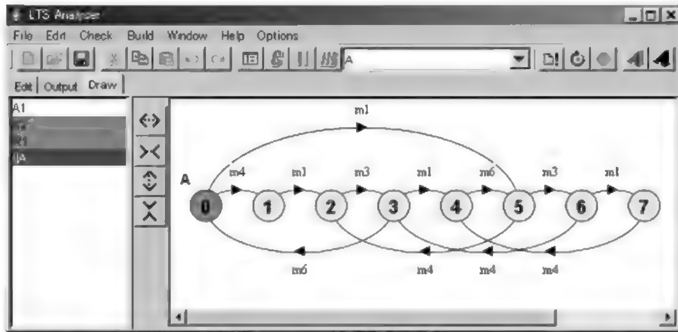


図10　ステート・マシンを自動合成したところ

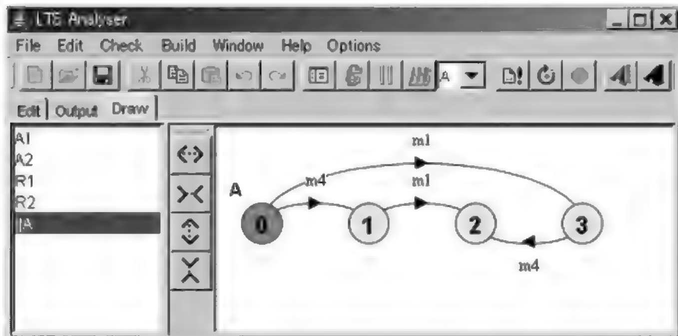


図11　デッドロック

プロセス代数を使用することでシナリオ・ベースでの繰り返し開発が式(7)のレベルで行えるようになるのである．

それだけではない，

制約2：m6を受信してからm2を送信する

のように，制約2をさらに追加してステート・マシンを生成するとどうなるだろうか．要求は自然言語のリストなので簡単に追加できるが，実際に動くものを作れるかどうかは保障されていない．制約1と制約2，それからシナリオ-1に含まれるm2->m3の順序制約を同時に満たすことは，じつはできない．LTSAツールで実際にステート・マシンを生成すると図11のようになり，デッドロックが検出される．m1とm4を処理したところまでしかステート・マシンは生成されない．

並列動作を含んだシステムの繰り返し開発では，単に繰り返しのオーバヘッドが問題になるだけでなく，新たに追加された要求によってデッドロックや特定のステート間でのみループするようなシステムができてしまう危険性がある．プロセス代数を使用することで検証もできるようになるのである．

まとめ

繰り返し開発にはどうしてもオーバヘッドがある．それは，シュワルツの不等式で表すことができる．このオーバヘッドは，おもにふるまいの合成部分に現れる．繰り返し開発では，アーキテクチャが重要だといわれるが，その意味が少し明らかになった．

> **Column　並列化オペレータ**
>
> 本文式(11)について説明する．右辺(A1||A2)の||を並列化オペレータ(parallel operator)と呼ぶ．プロセス代数では最初に出てくる重要な二項演算子である．二つのプロセス(A1とA2)を並列動作させたときのプロセスを表す．二つのステート・マシンやタスクを並列動作させたのと等価な一つのステート・マシンやタスクを生成する．
>
> 左辺の||Aの||は，LTSAツール内での表現でAが合成されたプロセスであることを表す単なる記号である．

繰り返してもオーバヘッドにならない部分がある．インターフェースと静的構造の追加である．アーキテクチャとは，インターフェースと構造にほかならない．開発の初期でアーキテクチャを安定化することができれば，オーバヘッドのかからない繰り返し開発が可能になる．

すると，オーバヘッドはふるまいの合成だけになる．ふるまい合成オーバヘッドは，避けることは難しい．しかし，軽減することはできる．それは，プロセス代数を使って自動合成する手法である．

今回は，前回，前々回のアクティブ・オブジェクト・モデリングの話から少しわき道にそれて，繰り返し開発のオーバヘッドを減らすためにどのようにアクティブ・オブジェクト・モデリングを応用するかについて説明した．

参考文献

(1) TRC MARC人名典拠録編集部，17人のスティーヴンソン，外国人名のカナ表記1，図書館流通センター，1985.
(2) Martin Fowler，リファクタリング プログラミングの体質改善テクニック，ピアソン，2001.
(3) LTSAツール，http://www.doc.ic.ac.uk/~jnm/book/ltsa/LTSA.html：LTSAツールはダウンロードできる並行プログラムを検証するためのツールである．

ふじくら・としゆき　組み込みコンサルタント
tfujikura@jcom.home.ne.jp

TMS320C6713搭載DSPスタータ・キットを使った

C++によるDSPオブジェクト指向プログラミング

第5回 FFTを利用するディジタル・フィルタのためのクラス

◆三上 直樹

今回は，FFT（高速フーリエ変換）を使ったディジタル・フィルタを実現するための準備として，三つのクラスを作成します．

FFTでディジタル・フィルタを実現する際には，実行効率を高めるために，実信号用のFFTを使います．そこで最初に，実信号のFFTについて説明し，そのプログラムを前回作成した基底クラスに対する派生クラスとして作成します．

次に，C++に組み込みの配列では，以降のプログラムを作成する際に不便なので，新たに汎用1次元配列を実現するクラスを作成します．このクラスは，異なったデータ型にも対応できるようにテンプレート・クラスとして作成します．

最後に，FFTを使ってFIR（Finite Impulse Respons）型のディジタル・フィルタを実現する方法について説明し，そのためのクラスを作成します．

1 実信号のFFTと逆FFT

私たちの身の周りにある多くの信号の値は，実数として表現できます．このような信号を実信号といいます．たとえば，人間の音声をマイクで拾った信号は実信号です．

前回作成したFFTクラスは，データとして複素数を扱うようになっていましたが，FFTで実信号を扱う場合には，信号をデータの実数部に代入し，データの虚数部を0にするだけで対応できます．しかし，扱う信号を実信号に限定すると，計算量を減らすことができるので，さらに高速なFFTプログラムを作成することができます．また，実信号のFFTをさらに逆FFT（IFFT）する場合も同様に高速化が図れます．

● FFTの場合

FFTアルゴリズムとして前回示した周波数間引きアルゴリズムを使う場合，第1段目と最後の2段の計算量を減らすことができます[注1]．

▶ 1段目の処理

FFTでの基本的な演算は，前回示したようにバタフライ演算です．その中の$B=(a-b)W_{16}^{k}$という演算について考えてみます．ここで，W_{16}^{k}は特定のkを除くと複素数になるので，aとbを複素数とすれば，バタフライ演算での乗算は，

複素数×複素数

という形になります．Re{ }を実数部を表す記号，Im{ }を虚数部を表す記号とし，この計算を実数部と虚数部に分けて書くと，次のようになります．

$$\mathrm{Re}\{B\}=\mathrm{Re}\{a-b\}\times\cos(2\pi k/16)-\mathrm{Im}\{a-b\}\times\sin(2\pi k/16)$$
$$\mathrm{Im}\{B\}=\mathrm{Re}\{a-b\}\times\sin(2\pi k/16)+\mathrm{Im}\{a-b\}\times\cos(2\pi k/16) \quad \cdots\cdots(1)$$

したがって，$B=(a-b)W_{16}^{k}$という演算を行うためには，実数の乗算が4回必要になります．

しかし，aとbが実数であれば，バタフライ演算での乗算は，

実数×複素数

という形になります．つまり，実数部と虚数部に分けて書くと，次のようになります．

$$\left.\begin{array}{l}\mathrm{Re}\{B\}=\mathrm{Re}\{a-b\}\times\cos(2\pi k/16)\\ \mathrm{Im}\{B\}=\mathrm{Re}\{a-b\}\times\sin(2\pi k/16)\end{array}\right. \quad \cdots\cdots(2)$$

つまり，実数の乗算が2回で済むことになります．

したがって，入力データが実数の場合，第1段目の処理では実数の乗算回数を半分に減らすことができます．加減算についても同様です．

▶ DFT（離散フーリエ変換）の性質

最後の2段の処理について説明する前に，その処理で使うDFTの性質について説明します．信号$g[n]$のDFTである$G[k]$の定義を式（3）に示します．

$$G[k]=\sum_{n=0}^{N-1}g[n]\exp\left(\frac{-j2\pi nk}{N}\right),\ k=0,1,\cdots,N-1 \quad \cdots(3)$$

信号$g[n]$が実数，つまり実信号の場合，そのDFTである$G[k]$は次の性質を持っています．

$$G[k]=G[N-k]^{*} \quad \cdots\cdots(4)$$

ここで，*は複素共役（complex conjugate）を表します．

式（4）の関係を$G[k]$の実数部と虚数部に分けて表すと，次の

注1：そのほかの段も，1や0を乗算する場合などのように，計算量を減らすことができる箇所が存在する．しかし，そのような箇所で計算量を減らすためには，`if`文などの条件文が必要になるので，かえって実行時間が増加してしまう場合もあるので，注意しなければならない．

関係が成り立ちます．

$$\mathrm{Re}\{G[k]\} = \mathrm{Re}\{G[N-k]\} \quad \cdots\cdots (5)$$

$$\mathrm{Im}\{G[k]\} = \mathrm{Im}\{G[N-k]\} \quad \cdots\cdots (6)$$

このようすを図1に示します．なお，$g[n]$が実信号の場合，$G[0]$と$G[N/2]$の虚数部は必ず0になります．

以上のことから実信号のDFTを求める場合，式(3)の計算を$k=0, 1, \cdots, N/2$について計算すれば十分で，$k=(N/2)+1, (N/2)+2, \cdots, N-1$については計算する必要のないことがわかります．

▶ 最後の2段の処理

次に，今説明したDFTの性質を使って最後の2段の処理での計算量を減らす方法を，図2を使って説明します．式(4)の性質から，図2(a)に示す最後の2段の処理で，B，Dについての計算は必要のないことがわかります．さらに，ここに現れる回転因子の値はすべて1か$-j$になります．そこで，この部分の処理は図2(b)のようになり，実際には乗算が不要になります．

以上のことをまとめ，さらに最後の2段の処理を1段で実行するように変形すると図2(c)のように書けることになります．

● 逆FFTの場合

ある実信号のDFTを$G[k]$とすると，$G[k]$に逆FFT処理を行うと実信号が得られます．一方，最終段のバタフライ演算は加算と減算だけで実行できます．したがって，最終段のバタフライ演算では，虚数部が必ず0になることがわかっているので虚数部に関する計算は不要で，実数部に関する計算のみでよいことになります．

2 実信号のFFTのためのクラス

実信号のFFTを行うためのクラス`RealFFT`および実信号のFFTの逆FFT(IFFT)を行うためのクラス`IFFTReal`をリスト1に示します．この二つのクラスは，前回のFFTの基底クラス(`BaseFFT`)を継承して作成します．

▶ ヘッダ(**`MyFFTReal.hpp`**)

リスト1(a)にヘッダを示します．どちらのクラスも，ヘッダで定義されているコンストラクタは，基底クラスにFFTのデータ数を渡し，基底クラスのコンストラクタを起動するだけです．ただし，クラス`IFFTReal`の場合は逆FFTを行うので，

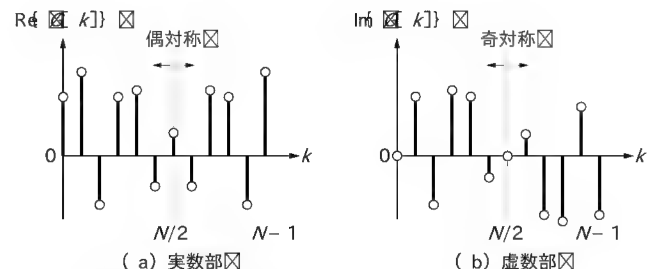


図1 実信号のDFTの実数部と虚数部のようす(Nが偶数の場合)

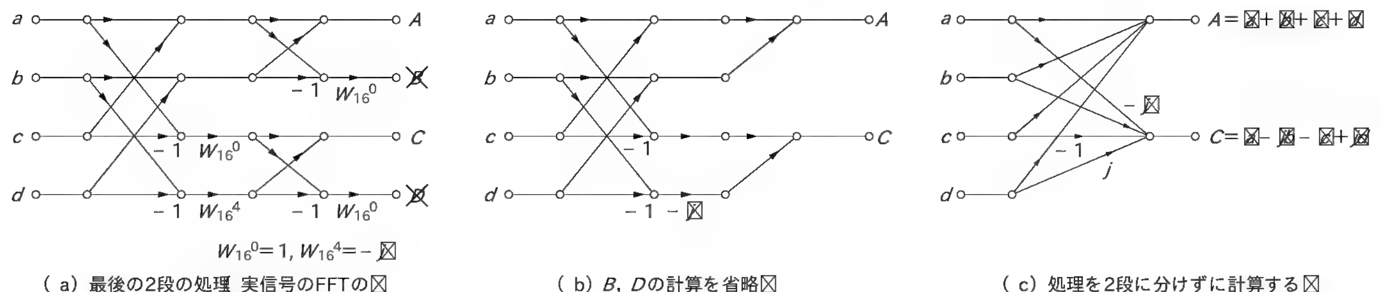


図2 実信号のFFTを行う場合の，最後の2段の処理の変型

リスト1 実信号のFFTのためのクラス

```
//----------------------------------------------
// BaseFFT の派生クラス(ヘッダ)
//      RealFFT    : 実信号に対する FFT
//      IFFTReal   : 実信号の FFT に対する逆 FFT
//----------------------------------------------
#ifndef MK_MyFFTReal

#include "MyFFTBase.hpp"

//----------------------------------------------
// 実信号に対する FFT
class RealFFT : public BaseFFT
{
public:
    RealFFT(int n) : BaseFFT(n) {}              // コンストラクタ
    void Execute(const float r[], Complex x[]);// FFT の実行
};

//----------------------------------------------
// 実信号の FFT に対する逆 FFT
class IFFTReal : public BaseFFT
{
public:
    IFFTReal(int n) : BaseFFT(-n) {}       // コンストラクタ
    void Execute(Complex x[], float r[]); // 逆 FFT の実行
};

#define MK_MyFFTReal
#endif
```

リスト 1　実信号の FFT のためのクラス(つづき)

```
//----------------------------------------------
// BaseFFT の派生クラス(定義本体)
//      RealFFT   : 実信号に対する FFT
//      IFFTReal  : 実信号の FFT に対する逆 FFT
//----------------------------------------------
#include "MyFFTReal.hpp"

//----------------------------------------------
// 実信号に対する FFT の実行
//----------------------------------------------
void RealFFT::Execute(const float r[], Complex x[])
{
    int knh;
    float xR, xI, xN2;

    n_half = n_abs >> 1;

// 第1 段目の処理
    for (int k=0; k<n_half; k++)
    {
        knh = k + n_half;
        x[k] = r[k] + r[knh];                   // バタフライ演算
        x[knh] = (r[k] - r[knh])*w_tbl[k];      // バタフライ演算
    }
    n_half = n_half >> 1;

// 第2，3，・・・・，第(log2(nFFT)-2) 段目の処理
    for (next=2; next<(n_abs>>2); next<<=1) FFT_Loop(x);

// 最後の2 段の処理
    xN2 = x[0].Real() - x[1].Real() + x[2].Real() - x[3].Real();
    for (int k=0; k<n_abs; k=k+4)
    {
        xtmp = x[k] + x[k+1] + x[k+2] + x[k+3];
        xR = x[k].Real() + x[k+1].Imag() - x[k+2].Real()
                                         - x[k+3].Imag();
        xI = x[k].Imag() - x[k+1].Real() - x[k+2].Imag()
                                         + x[k+3].Real();
        x[k+2] = Complex(xR, xI);
        x[k] = xtmp;
    }
    x[1] = Complex(xN2, 0.0);
// ビット逆順の並べ替え
    for (int k=0; k<(n_abs>>1); k++)
        if (k < b_tbl[k]) Swap(x[k], x[b_tbl[k]]);
}

//----------------------------------------------
// 実信号の FFT に対する逆 FFT の実行
//----------------------------------------------
void IFFTReal::Execute(Complex x[], float r[])
{
    n_half = n_abs >> 1;

// 複素共役に対する処理
    for (int k=1; k<n_half; k++) x[n_abs-k] = Conj(x[k]);

// 第1，2，・・・・，第(log2(nFFT)-2) 段目の処理
    for (next=1; next<(n_abs>>1); next<<=1) FFT_Loop(x);

// 最終段の処理，ビット逆順の並べ替え，1/N の乗算
    for (int k=0; k<n_abs; k=k+2)
    {
        r[b_tbl[k]] = nInv*(x[k].Real() + x[k+1].Real());
        r[b_tbl[k+1]] = nInv*(x[k].Real() - x[k+1].Real());
    }
}
```

(b) MyFFTReal.cpp

データ数に“-”を付けて基底クラスに渡しています．

▶ メンバ関数の定義(**MyFFTReal.cpp**)

リスト1(b)に FFT および逆 FFT 実行のためのクラスのメンバ関数 Execute() の定義を示します．このメンバ関数は，どちらの場合も第1引き数が入力データ，第2引き数が計算結果になっています．

FFT のためのクラス **RealFFT**

FFT の処理の部分は，第1段目，最後の2段を除く第2段目以降，最後の2段の三つに分かれています．

第1段目の処理では，バタフライ演算が実数と複素数の乗算で実行できるので，実数の乗算回数が半分になります．

最後の2段以外の第2段目以降の処理は，通常のバタフライ演算になるので，基底クラスのメンバ関数 FFT_Loop() を使ってバタフライ演算を行っています．

最後の2段の処理は図2(c)に従って処理を行います．ただし，$C[N/2]$(N：データ数)の値は，図2(c)の処理では計算できないので，この値は別に計算します．

最後に，ビット逆順の並べ替えを行います．

処理の結果は，メンバ関数 Execute() の第2引き数 x[] に与えられます．この配列のデータの中で，x[0]～x[N/2](N：FFT の点数)が有効なデータです．x[(N/2)+1]～x[N-1] に得られるデータは計算の途中結果で，無効なデータです．

逆 FFT のためのクラス **IFFTReal**

逆 FFT の処理も大きく三つに分けることができます．

最初は，式(3)の関係を使い，$C[k]$，k=1，2，…，$(N/2)-1$ に対応するデータの共役複素数 $C[k]^*$ を，$C[N-k]$，k=1，2，…，$(N/2)-1$ に対応する箇所に格納します．したがって，Execute() の第1引き数には，$C[k]$の中で，k=1，2，…，$N/2$ に対応するデータのみを与えればよいということになります．

次に，最後の1段以外の処理を行います．この部分は通常のバタフライ演算になるので，基底クラスのメンバ関数 FFT_Loop() を使ってバタフライ演算を行っています．

最後の段では，バタフライ演算とビット逆順の並べ替えの乗算を一つのループ処理の中でまとめて行っています．この段のバタフライ演算は加算と減算だけで実行でき，しかも最終的な結果の虚数部が0になるということがわかっています．そのため，バタフライ演算では入力の実数部だけを使って計算を行っています．

最後の段はほかの段と異なり，バタフライ演算を行う際の入力データが格納されている配列と，出力データを格納する配列は別なものとなっています．そこで格納する際に，ビット逆順の位置に格納するということが可能です．また格納の際に，$1/N$ の乗算もあわせて行っています．

処理の結果は，メンバ関数 Execute() の第2引き数 r[] に与えられます．

3　汎用1次元配列クラス

配列はプログラムを作るうえで欠かせないものです．ところ

リスト 2　汎用 1 次元配列クラス(MyArray.hpp)

```
//-----------------------------------------------------------------
// 汎用1 次元配列クラス
// 作成者:  三上直樹, 2003/11/14
//-----------------------------------------------------------------

#ifndef MK_MyArray

#include <cassert>// assert() で使用
#include <new>    // new, delete, set_new_handler() で使用
using namespace std;

template <class T> class Array
{
private:
    T    *v;
    int  size;
    void Range(int pos) const;        // 添え字の範囲チェック
    void Copy(const Array<T>& v_src);
                                      // Arrayオブジェクトのコピー
    inline void ArrayNew(const int i);
                                      // コンストラクタで使うルーチン
public:
    explicit Array(int i = 1) { ArrayNew(i); }
                                      // コンストラクタ
    Array(const Array<T>& a) { Copy(a); }
                                      // コピー・コンストラクタ
    inline Array(int i, T init_v);
                              // 配列の内容を初期化するコンストラクタ
    ~Array() { delete[] v; }          // デストラクタ
    void SetSize(int i);              // 配列の大きさの再設定
    Array<T>& operator=(const Array<T>& a);
                                      // “=”演算子
    inline T& operator[](int i);      // “[]”演算子, 左辺で使用
    inline const T& operator[](int i) const;
                                      // “[]”演算子, 右辺で使用
    operator const T* () const { return v; }
                                         // 型変換用演算子
    operator T* () const { return v; }     // 型変換用演算子
};

// 汎用1 次元配列クラスの定義部

template <class T> void Array<T>::Range(int pos) const
{
    assert(0 <= pos && pos < size);   // 添え字が範囲外か?
}

template <class T> void Array<T>::Copy(const Array<T>& v_src)
{
    v = new T[size = v_src.size];
    assert(v);                        // メモリ確保のエラー
    for (int i=0; i<size; i++) v[i] = v_src.v[i];
}

// コンストラクタで使うルーチン
template <class T> void Array<T>::ArrayNew(int i)
{
    set_new_handler(0);
    assert(i > 0);                      // サイズ指定のエラー
    v = new T[size = i];
    assert(v);                          // メモリ確保のエラー
}

// 配列の内容を初期化するコンストラクタ
template <class T> inline Array<T>::Array(int i, T init_v)
{
    ArrayNew(i);
    for (int n=0; n<i; n++) v[n] = init_v;
}

template <class T> void Array<T>::SetSize(int i)
{
    delete[] v;
    assert(i > 0);                      // サイズ指定のエラー
    v = new T[size = i];
    assert(v);                          // メモリ確保のエラー
}

template <class T> Array<T>& Array<T>::operator=
                                            (const Array<T>& a)
{
    if (this != &a)                     // 自己代入の禁止
    {
        delete [] v;
        Copy(a);
    }
    return *this;
}

template <class T> inline T& Array<T>::operator[](int i)
{
#ifdef DEBUG_ARRAY_CHECK
    Range(i);                           // 添え字が範囲外か?
#endif
    return v[i];
}

template <class T> inline const T& Array<T>::operator[](int i)
                                                           const
{
#ifdef DEBUG_ARRAY_CHECK
    Range(i);                           // 添え字が範囲外か?
#endif
    return v[i];
}

#define MK_MyArray
#endif
```

が，C++でいろいろなプログラムを作成してみると，C++が最初から備えている配列では何かと使いにくい点が出てきます[注2]．そこで，以降のプログラムで使用するため，クラスを使って改良された配列を作成します．

新たに作成するクラスは，1次元の配列とし，配列に含まれる要素のデータ型が何であっても使えるような汎用クラスにするため，テンプレート・クラス(template class)として実現します．なお，テンプレート・クラスと似たものとして，C++の便利な機能の一つにテンプレート関数というものがあります．これについては，コラムを参照してください．

リスト 2(MyArray.hpp)に汎用1次元クラス Arrayを示します．

● テンプレート・クラスの書きかた

テンプレート・クラスを宣言する場合は，先頭に templateというキーワードを付け，それに続けて <>内に classというキーワードとデータの型に対応するパラメータを書きます．リスト 2では，このパラメータとして Tと書いています．この名前は，通常の変数名として使えるものであれば，何でもかまいません．テンプレート・クラスの内部では，このパラメータをデータ型の名前のように扱ってプログラムを記述します．

注 2：C++の組み込みの配列にはいくつかの欠点がある．大きな欠点として，次の二つが挙げられる．多くの処理系では，配列の大きさはコンパイル時に決められていることが必要で，実行時に配列の大きさを変化させることはできない．配列の範囲外をアクセスするような添え字を使っても，それをチェックする機能はない．

● 非公開メンバ

クラスArrayは非公開のデータ・メンバを二つ持ちます．vはTという型のデータからなる配列の先頭の要素を指すポインタとして使います．このTに対応する型は，テンプレート・クラスのオブジェクトを宣言する際に決まり，たとえばint型だったり，float型だったり，あるいはユーザの定義したクラス名です．sizeは配列のサイズです．

そのほか，非公開のメンバ関数として，Range()，Copy()，ArrayNew()が宣言されています．

メンバ関数Range()は，関数assert()を使って配列の添え字の範囲をチェックします．このRange()は，演算子"[]"を実現する関数の中で使われます．

メンバ関数Copy()は，Arrayオブジェクトのための領域を確保し，その領域へすでに存在するArrayオブジェクトの内容をコピーします．このCopy()は，コピー・コンストラクタと演算子"="を実現するメンバ関数の中で使われます．

メンバ関数ArrayNew()は，いくつかのコンストラクタで共通に行われる処理をまとめたものです．この関数では，最初に実行される，

```
set_new_handler(0);
```
注3

の処理により，以降で使われるnew演算子による領域確保に失敗した場合に，NULLポインタが返されるように設定しています．

● 公開メンバ

コンストラクタとしては，デフォルト・コンストラクタ，コピー・コンストラクタ，配列の内容を指定された値に初期化するコンストラクタの3種類が宣言されています．

デフォルト・コンストラクタは，explicitの指定がなされていますが，これは暗黙の型変換を防止するためのものです．

メンバ関数SetSize()は，すでに存在するArrayオブジェクトのサイズを変更します．

operator=()は，Arrayオブジェクトをコピーするための演算子"="を実現するメンバ関数です．

配列の要素にアクセスする手段を提供するのが，演算子"[]"です．この演算子を実現するメンバ関数operator[]()については，非constバージョンとconstバージョンの2種類を用意しました．非constバージョンを用意した理由は，左辺値として使えるようにするためです．この二つのメンバ関数operator[]()は，いずれも関数Range()により，添え字の範囲をチェックするようにしています．ただし，これは実行効率を悪化させるので，#ifdefと#endifによって条件コンパイルされるようにしてあり，通常はこのチェックは行わないようにしています．範囲のチェックを有効にしたい場合には，事前に"DEBUG_ARRAY_CHECK"を定義する必要があります．

operator T*()は型変換（キャスト）のためのメンバ関数で，これも非constバージョンとconstバージョンの2種類を用意しました．

● 汎用1次元クラスの使いかた

このクラスを使って，float型のデータを要素とし，サイズが100で，xという名前の配列を宣言する場合は，次のように記述します．

```
Array<float> x(100);
```

<>内には，floatのような，C++に組み込みのデータ型だけでなく，ユーザの定義したクラス名なども書くことができます．

この汎用1次元クラスは，#define文でユーザが明示的に"DEBUG_ARRAY_CHECK"を定義しておくと，配列の要素にアクセスする際に，添え字の範囲を調べ，範囲外であればメッセージを出して異常終了するようになっています．ただし，この場合は実行効率が悪くなります．この定義を行わなければ，添え字の範囲のチェックは行わないので，実行効率は悪くなりません．

4 FFTによるFIRフィルタ

FFTを使うと，FIRフィルタの計算を効率良く行うことができます．そこで，最初に信号の長さが有限の場合の方法について簡単に説明します．次にその方法を，信号の長さが無限の場合（非常に長い場合）に適用する方法について説明します．

● 信号の長さが有限の場合

FIRフィルタの係数を$h[m]$（m=0，1，…，$N-1$）とすると，そのフィルタの入力$x[n]$と出力$y[n]$の関係は次の式で表現できます[注4]．

$$y[n]=\sum_{m=0}^{N-1}h[m]\cdot x[n-m] \quad \cdots\cdots (7)$$

このような式は，一般に畳（たた）み込み（convolution）と呼ばれています．この畳み込みは，次に説明する循環畳み込みと区別するため，直線畳み込み（linear convolution）とも呼ばれています．

一方，DFTの性質の一つに，循環畳み込み（circular convolution）定理があります．それは次のようなものです．まず，$h[m]$と$x[n]$との循環畳み込みは次の式で表されます．

$$y[n]=\sum_{m=0}^{N-1}h[m]\cdot x[(n-m)\bmod N] \quad ^{注5} \cdots\cdots (8)$$

このとき，$y[n]$，$h[m]$，$x[n]$，（n=0，1，…，$N-1$）のN点DFTをそれぞれ$Y[k]$，$H[k]$，$X[k]$とすると，次の式が成り立ちます．

注3：関数set_new_handler()については，前回のコラムを参照．

注4：正確にいうと，そのディジタル・フィルタが因果的である場合．

注5：整数n，n_1，n_2に対して，$n=n_1+N\cdot n_2$の関係があるものとする．ただし，$0\leq n_1\leq N-1$とする．このとき，整数n_1は$n_1=n \bmod N$または$N_1=n$ modulo Nのように表すことができる．したがって，nが正の場合，$n_1=n \bmod N$で記述されるn_1は，nをNで割り算したときの余りと考えることができる．

Column テンプレートでプログラムを簡潔に

Cでは，データの型は違うが同じ処理を行うような関数を作りたい場合，`#define`によるマクロを使うと，一通りの定義で済ませることができます．しかし，マクロには副作用があったり，コンパイルの際に型をチェックしないといった問題があります．C++ではテンプレート(template)が新たに使えるようになり，そのような問題を解決することができます．なお，テンプレートはオブジェクト指向とは別の面での，Cの強化策の一つです．

テンプレートには，テンプレート関数とテンプレート・クラスがあります．テンプレート・クラスについては本文中で説明しているので，ここではテンプレート関数について説明します．

たとえば，二つの変数の値を交換する`Swap()`という関数をテンプレート関数として書くと，次のようになります．なお，`template`と`class`はキーワードなので，太字にしています．

```
template <class Type> void Swap(Type &a, Type &b)
{
    Type tmp = a;
    a = b;
    b = tmp;
}
```

テンプレート関数を宣言するときは，先頭にキーワード`template`を付け，続いて`<>`内にキーワード`class`とデータの型に対応するパラメータ`Type`を書きます．ここまではテンプレート・クラスと同じです．それ以降は通常の関数と同じ書きかたになります．`<>`内の`Type`は，データの型に対応する仮のパラメータで，変数名として許される名前であれば何を書いてもかまわないという点もテンプレート・クラスと同じです．

このテンプレート関数を使うと，次のプログラムのように，引き数のデータの型が違っても定義が一つで済むので，プログラムの開発が楽になります．なお，`Complex`は前回説明した複素数クラスです．

```
int main()
{
    int a1 = 1, a2 = -2;
    Swap(a1, a2);
    printf("a1 =    %d, a2 =     %d¥n", a1, a2);

    char b1 = 'A', b2 = 'B';
    Swap(b1, b2);
    printf("b1 =   '%c', b2 =   '%c'¥n", b1, b2);

    Complex c1(5, 6), c2(7, 8);
    Swap(c1, c2);
    printf("c1 = (%1.0f,%1.0f),
            c2 = (%1.0f,%1.0f)¥n",
            c1.Real(), c1.Imag(),
            c2.Real(), c2.Imag());

    return 0;
}
```

このプログラムを実行すると次のような結果が得られます．

```
a1 =    -2 , a2 =     1
b1 =   'B' , b2 =   'A'
c1 = (7,8) , c2 = (5,6)
```

$$Y[k] = H[k] \cdot X[k] \quad \cdots\cdots (9)$$

これを循環畳み込み定理といいます．そこで，逆FFTの操作をIFFT{}で表すものとすると，$h[m]$と$x[n]$との循環畳み込みは，次のようにして計算できることになります．

$$y[n] = \mathrm{IFFT}\{H[k] \cdot X[k]\} \quad \cdots\cdots (10)$$

このようにFFTを使って畳み込みの計算を行っても，まわりくどいだけで何もメリットがないように思うかもしれません．しかし，データ数Nがある程度以上に大きい場合は，式(8)を直接計算するよりも，FFTを使ったほうが計算量はかなり減ります．その理由は，FFTを使うと計算量が$N\log N$に比例するのに対して，式(8)で直接計算すると計算量がN^2に比例するようになるからです．

ところで，式(10)の結果は循環畳み込みになるので，式(7)の直線畳み込みとは異なるものです．そこで，フィルタの係数$h[m]$の個数Nと，入力信号$x[n]$の個数をLとします．そして，$h[m]$，$x[n]$のDFTをM点FFTで計算し[注6]，同様に式(10)の計算にもM点IFFTを使うものとします．このとき，次の式(11)が成り立つ場合に，式(10)で計算した$y[n]$は，式(7)で計算した直線畳み込み$y[n]$に一致します．

$$N + L - 1 \leqq M \quad \cdots\cdots (11)$$

● 信号の長さが無限の場合

現実の信号に対してFIRフィルタの処理をリアルタイムで行わせようとする場合，一般に信号の長さは決まっていません．そのような場合は，信号の長さを無限とみなして処理を行う必要があります．そのため，前の項で説明した方法をそのまま使うことはできません．このような場合は，信号をブロックに分割したうえで，前項で説明した方法を適用します．そのための

注6：FFTを行う場合は，データの不足する部分に0(ゼロ)を追加する．つまり，フィルタの係数$h[m]$の後には$M-N$個の0を追加し，入力信号$x[n]$の後には$M-L$個の0を追加してからFFTを行う．

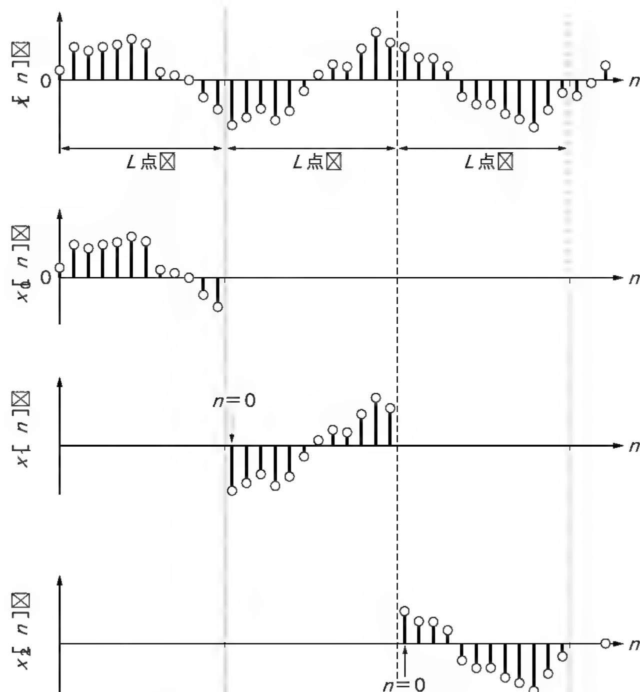


図3　重複加算法で入力信号を分割するようす
Lは1ブロックの個数．L=M−N+1，M，Nについては，図4を参照のこと．

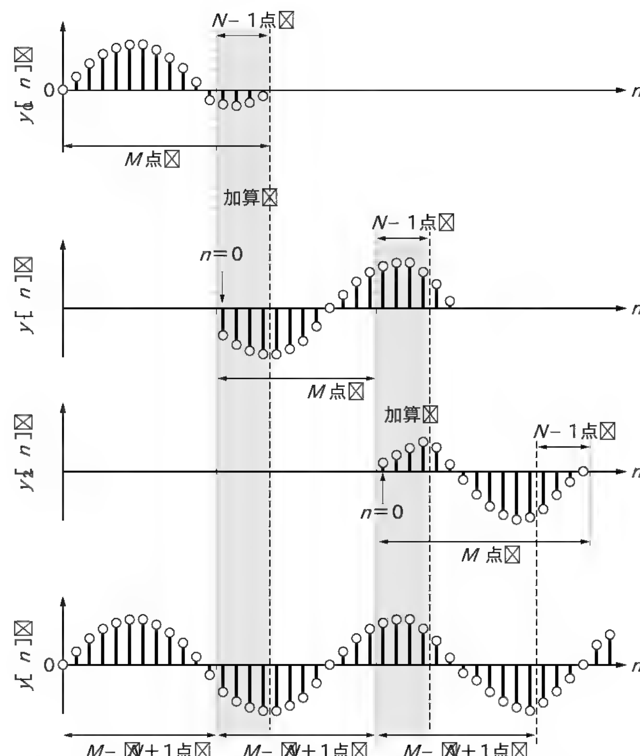


図4　重複加算法で出力信号を得るための処理のようす
網掛け部は加算を行う部分を示す．Nはフィルタの係数の個数を，Mは使用するFFTの点数を示す．

方法として，重複加算法（overlap-add method）と重複保持法（overlap-save method）という二つの方法が知られています[1,2]．

▶ 重複加算法の原理

最初に入力信号$x[n]$をブロックに分割します．第m番目のブロックを$x_m[n]$とし，1ブロックのデータ数をLとすると，次のようになります．

$$x_m[n]=\begin{cases} x[n+mL], & 0\le n\le L-1 \\ 0, & それ以外 \end{cases} \quad \cdots\cdots (12)$$

ただし，$x_m[n]$の時間の原点は，$x[n]$の時間の原点ではなく，各ブロックの先頭にあるデータの位置を原点としています．このブロックに分割したようすを図3に示します．この図には，各ブロックの原点をn=0というぐあいに表しています．$x_0[n]$，$x_1[n]$，$x_2[n]$はブロックに分割された信号です．この$x_m[n]$を使うと$x[n]$は次のように表すことができます．

$$x[n]=\sum_{m=0}^{\infty} x_m[n-mL] \quad \cdots\cdots (13)$$

一方，畳み込みの記号を$*$とすると，入力信号の第m番目のブロック$x_m[n]$とフィルタの係数$h[n]$との畳み込みを行った結果から得られる$y_m[n]$は次のように表現できます．

$$y_m[n]=h[n]*x_m[n] \quad \cdots\cdots (14)$$

したがって，出力信号$y[n]$は次のようにして求めることができます．

$$y[n]=h[n]*x[n]=h[n]*\sum_{m=0}^{\infty} x_m[n-mL]$$
$$=\sum_{m=0}^{\infty}(h[n]*x_m[n-mL])=\sum_{m=0}^{\infty} y_m[n-mL] \quad \cdots\cdots (15)$$

つまり，$h[n]$と第m番目のブロックの信号$x_m[n]$に対する畳み込みである$y_m[n]$をFFTを使って求め，式(15)のようにその結果の和を求めれば，入力信号$x[n]$とフィルタ係数$h[m]$の畳み込みを求めることができます．もちろんこのときに，FFTを使って畳み込みの計算を行うので，直線畳み込みとなる条件である式(11)の関係を満足しなければなりません．そこで，フィルタの係数の個数をNとすると，FFTの点数Mは通常次のように設定します．

$$M=N+L-1 \quad \cdots\cdots (16)$$

この場合，式(15)の計算を行うと，第mブロックの信号に対する畳み込みの後の$N-1$個のデータと，第$m+1$ブロックの信号に対する畳み込みの先頭から$N-1$個のデータが重複します．そこでこの重複する部分は，加え合わせたものを出力とします．出力のところで行う処理のようすを図4に示します．

▶ 重複保持法の原理

重複保持法では，入力信号をブロックに分割しますが，一部の入力データを重複して用います．このとき，FFTを使って行う畳み込みが（循環畳み込みではなく）直線畳み込みになる条件

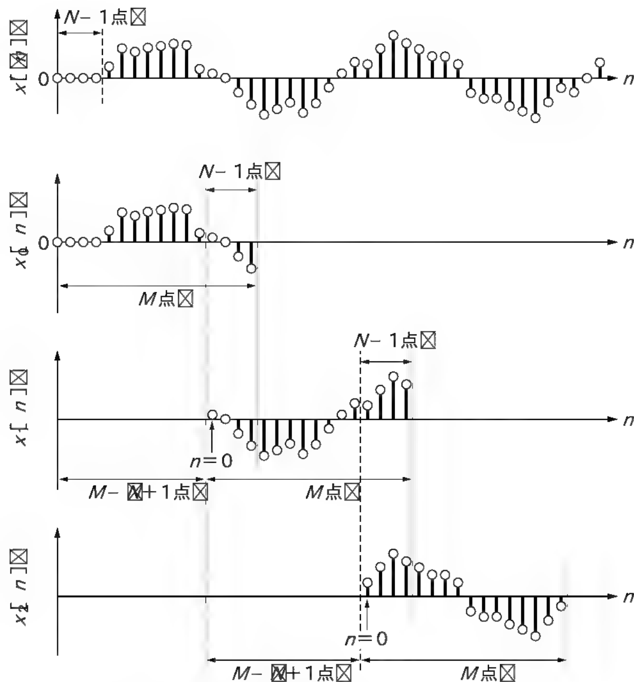


図5　重複保持法で入力信号を分割するようす
Nはフィルタの係数の個数を，Mは使用するFFTの点数を示す．

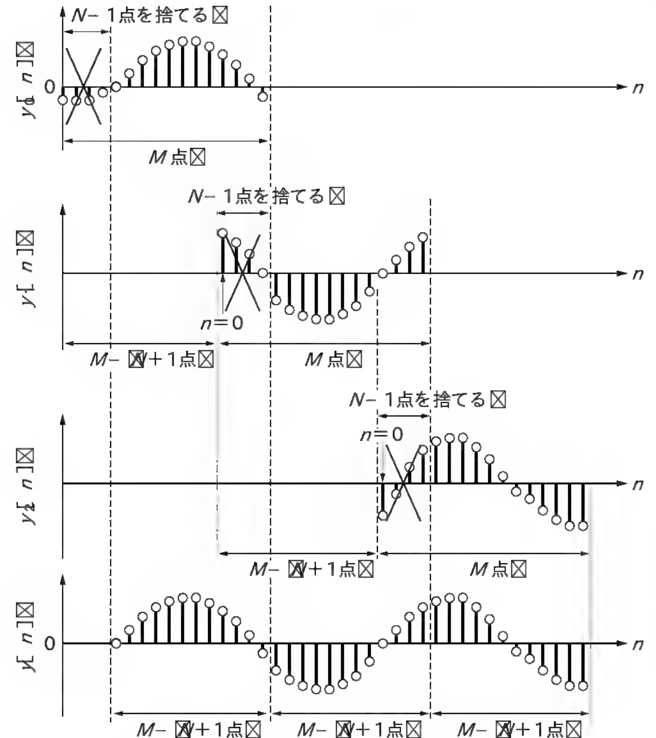


図6　重複保持法で出力信号を得るための処理のようす
Nはフィルタの係数の個数を，Mは使用するFFTの点数を示す．

である式(11)を考慮に入れて考えてみます．

入力信号の個数と使用するFFTの点数が等しい(式(11)で $L=M$)の場合には，次のように考えることができます．つまり，式(10)で計算された出力信号の中で，先頭から $N-1$ のデータは循環畳み込みの結果であり，残り $M-N+1$ 個のデータは直線畳み込みの結果に一致すると考えることができます．したがって式(10)を使って計算されたデータの中で，先頭の $N-1$ 個のデータを捨ててしまえば，畳み込みの結果は循環畳み込みではなく，直線畳み込みになります．

以上のことから，入力信号をブロックに分割する際には1ブロックのデータ数をFFTの点数と等しい M 個とし，次のブロックとは $N-1$ 個のデータを重複させればよいことがわかります．したがって，データを取得するブロックの範囲を $M-N+1$ 個分だけシフトしながら処理を行っていくことになります．なお，最初のブロックだけはその一つ前のブロックのデータが存在しないので，先頭に $N-1$ 個の0(零値)を付加します．つまり，フィルタ処理を行う前の元の信号 $x[n]$ を $N-1$ 個だけ後にシフトし，先頭に0を追加したものを $x'[n]$ とすると，$x'[n]$ とブロックに分割したときの第 m ブロックの信号 $x_m[n]$ の関係は次のようになります．

$$x_m[n]=\begin{cases} x'[n+m(M-N+1)], & 0 \le n \le M-1 \\ 0 & ,(それ以外) \end{cases} \quad \cdots\cdots(17)$$

このようすを図5に示します．この図で，各ブロックの時間の原点は $n=0$ というぐあいに示しています．$x_0[n]$，$x_1[n]$，$x_2[n]$ はブロックに分割された信号です．この図から，第 m ブロックの後部 $N-1$ 個のデータと，第 $m+1$ ブロックの先頭 $N-1$ 個のデータが重複しているようすがわかります．

出力する際は，ブロックごとに行った畳み込みの計算結果の中で，先頭の $N-1$ サンプルを捨てながら一つの信号に合成していけば，その全体に対して直線畳み込みを行った結果，つまりフィルタの出力 $y[n]$ が得られます．

FFTを使って計算した第 m ブロックの出力信号から先頭の $N-1$ サンプルを除いた信号を $y_m[n]$ とすると，出力信号 $y[n]$ は次のように表されることになります．

$$y[n]=\sum_{m=0}^{\infty} y_m[n-m(M-N+1)],\ n=0,1,2,\cdots \quad \cdots\cdots(18)$$

このようすを図6に示します．この図で，各ブロックの時間の原点は $n=0$ というぐあいに示しています．

5 FFTにより畳み込みを行うクラス

ディジタル・フィルタをFFTで実現する際には，次の処理が必要になります．

1) FFTを利用して係数のDFTの計算を行う
2) 入力信号のDFTをFFTで求め，それと係数のDFTとの乗算を行い，さらに逆FFTを行う

これらのことを行うクラスが `ConvolverFFT` で，これをリスト3(`ConvolverFFT.cpp`)に示します．このクラスは，

リスト 3　FFTにより畳み込みを行うための基底クラス（ConvolveFFT.cpp）

```
//---------------------------------------------------------------
// FFTを使って畳み込みを行うためのクラス
//  作成者:  三上直樹, 2004
//---------------------------------------------------------------
#ifndef MK_ConvolverFFT

#include "MyFFTReal.hpp"
#include "MyArray.hpp"

// クラスConvolverFFTの宣言部
class ConvolverFFT
{
private:
    RealFFT   RFFT_M;   // 実データのFFTを実行するオブジェクト
    IFFTReal  IFFTR_M;  // 実データのFFTの逆FFTを実行するオブジェクト
    Array<Complex>Hk;   // フィルタ係数のDFTが格納される配列
    Array<Complex>Xk;   // 循環畳み込みのための作業用配列
protected:
    const int N;        // フィルタ係数の数
    const int M;        // 使用するFFTの点数
    const int N21p;     // M - N + 1
    Array<float>  tmp;  // 作業用配列
public:
    ConvolverFFT(const int nCoefs, const int nFFT,
                                 const float hm[]);
    virtual ~ConvolverFFT() {}
    void Execute(const float xn[], float yn[]);
};

// クラスConvolverFFTの定義部
// コンストラクタ
ConvolverFFT::ConvolverFFT(const int nCoefs, const int nFFT,
                                             const float hm[])
        : RFFT_M(nFFT), IFFTR_M(nFFT), Hk(nFFT), Xk(nFFT),
          tmp(nFFT), N(nCoefs), M(nFFT), N21p(nFFT-nCoefs+1)
{
// FFTを使ってフィルタ係数のDFTを計算する
    for (int n=0; n<N; n++) tmp[n] = hm[n];
    for (int n=N; n<M; n++) tmp[n] = 0.0;
    RFFT_M.Execute(tmp, Hk);
}

// FFTを使って循環畳み込みの計算を行う
void ConvolverFFT::Execute(const float xn[], float yn[])
{
    RFFT_M.Execute(xn, Xk);
    for (int k=0; k<=(M>>1); k++) Xk[k] = Xk[k]*Hk[k];
    IFFTR_M.Execute(Xk, yn);
}

#define MK_ConvolverFFT
#endif
```

フィルタの係数（☒点）☒　後に0を詰める☒

h_0, ☒, ☒, ☒…☒ h_{N-1}, 0, 0, ☒…☒, 0

FFTの点数＝☒

図 7　クラス RealFFT() のメンバ関数 Execute() に与えるフィルタの係数のようす

重複加算法と重複保持法のどちらでも使うことができます．

コンストラクタでは，最初にメンバ初期設定の機能を使っていくつかの設定を行います．まず，クラス RealFFT とクラス IFFTReal のコンストラクタへ FFT の点数を渡し，これらのコンストラクタを起動します．次に，クラス Array のオブジェクトとして宣言されている Hk，Xk，tmp の領域を確保します．最後に，const データ・メンバ N，M，N21p に値を設定します．ここまでがメンバ初期設定による処理です．

次に，実数データ用の FFT を用いて，与えられた FIR フィルタの係数の DFT を計算します．このとき，N をフィルタの係数の個数，M を使用する FFT の点数とすると，$N < M$ になります．そのため，FFT 処理のためのクラス RealFFT のメンバ関数 Execute() にフィルタの係数を渡す際にデータが不足します．そこで，図 7 に示すように，係数の後には 0 を付加してデータ数が M 個になるようにして，これを渡します．

デストラクタは仮想デストラクタとして定義されていますが，このデストラクタ自身はリストからもわかるように何も行いません．しかし，クラス ConvolverFFT を継承する派生クラスの中にもデストラクタが定義されている場合や，その派生クラスのメンバとして包含されているクラスの中でデストラクタが定義されている場合，このように仮想デストラクタを定義する必要があります．仮想デストラクタを定義していない場合は，メモリのリークが発生することがあります．その理由については本連載の第 7 回目で説明する予定です．

FFT を利用した畳み込みの計算はメンバ関数 Execute() が行います．この関数では，与えられた入力信号に対して FFT を使って計算します．次にこの結果と，すでに計算してある FIR フィルタの係数の DFT との乗算を行います．最後に，その逆 FFT を行います．

この処理の中で注意しなければならない点は，使用する FFT の点数を M とすると，乗算を行う for ループの繰り返しが M 回ではなく（$M/2$）＋ 1 回になっているということです．その理由は，逆 FFT のためのクラス IFFTReal の項で説明しているように，このクラス IFFTReal で，逆 FFT を実行するメンバ関数 Execute() は，先頭の（$M/2$）＋ 1 個のデータのみを使うようになっているからです．

＊　＊

次回は，今回作成したクラス ConvolverFFT を使って，FFT による FIR フィルタのプログラムを作成します．

参考文献

（1）E. Oran Brigham；“*The fast Fourier transform*”，Chapter 13，Prentice-Hall，1974年

（2）佐川，貴家；「高速フーリエ変換とその応用」第5章，昭晃堂，1993年

みかみ・なおき　職業能力開発総合大学校　情報工学科

オープンソースのITRON仕様OS

TOPPERSで学ぶRTOS技術

第7回 サービス・コールの概要・その4

岸田 昌巳

前回に引き続き，シミュレーション環境を使いながら各サービス・コールについて説明します．

同期・通信機能のサービス・コール

前回説明しましたが，同期・通信機能としては，セマフォ，イベント・フラグ，データ・キュー，メール・ボックスの機能があり，これらは同期機能と通信機能を別々に提供しているのではなく，ほぼ，どちらの用途にも使える機能が提供されています．

今回はメール・ボックスに関して解説します．表1にメール・ボックスに関してサポートしている機能の一覧を示します．

メール・ボックス

● メール・ボックスの概要

メール・ボックスの機能は，メッセージを実行中のタスクからほかのタスクに送付する機能です．メッセージとは，送りたいデータをタスク間で共有可能なメモリ上に書いたもののことをいいます．μITRON4.0仕様のカーネルでは，このメッセージをあつかうためのサービス・コールとしてメール・ボックスを用意しています．

このメール・ボックスにより，メッセージを使い，同期や通信を行うことができます．この同期や通信の相手は，タスク間を対象としています．

このメッセージを書くためのメモリ領域はアプリケーションで確保します．メモリを確保する方法は任意ですが，TOPPERS/JSPではメモリ領域を確保するためのサービス・コールも用意されており，通常はこのサービス・コールを使用します．このメモリ領域を管理する機能をメモリ・プール管理機能といいます．なお，サービス・コールの説明の順序から，このメモリ・プール管理機能で用意しているサービス・コールは次回に説明します．

ここまでは機能の説明でしたが，メール・ボックスの機能を中身から見ると，もう少しわかりやすくなるかと思います．処理の流れとしては，図1のようになっています．

● タスク例外などの処理から

メール・ボックスの機能では，タスク間を対象としているため，タスク以外の処理で使用する`ixxx_yyy`のサービス・コールは用意されていません．タスク外で使用可能なサービス・コールを表2に示します．つまり，ほかのタスク管理機能，同期・通信機能では，タスク間のやりとりだけでなく，タスク以外からもイベントをタスクに送ることができましたが，メール・ボックスではできないということを示しています．

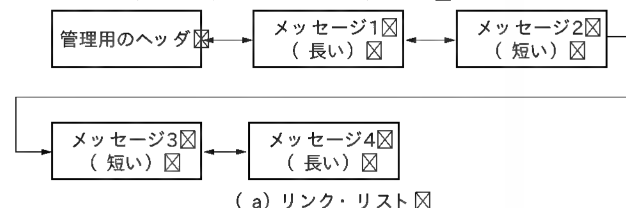
● メール・ボックス使用の勧め

TOPPERS/JSPカーネルでは，同期や通信を行うためのサービス・コールには，セマフォ，イベント・フラグ，データ・キュー，メール・ボックスがあります．あと，同期だけですが，タスクとの同期を取るためには，もう少しローレベルなタスク付属同期機能などもあります．

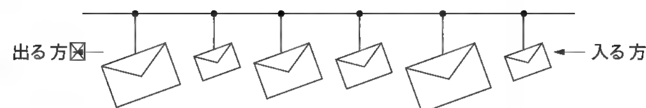
いろいろ用意されているので，適材適所で使うと言いたいと

表1 メール・ボックス機能のサービス・コール一覧

メールボックス
ER snd_mbx(ID mbxid, T_MSG *pk_msg);
ER rcv_mbx(ID mbxid, T_MSG **ppk_msg);
ER prcv_mbx(ID mbxid, T_MSG **ppk_msg);
ER trcv_mbx(ID mbxid, T_MSG **ppk_msg, TMO tmout);



（a）リンク・リスト



（b）模式的な図

図1 メール・ボックスの処理の流れ

表2　タスク外で使用可能なサービス・コール

タスク管理機能	
iact_tsk	タスクの起動
タスク付属同期機能	
iwup_tsk	タスクの起床
irel_wai	待ち状態の強制解除
タスク例外処理機能	
iras_tex	タスク例外処理の要求
同期通信機能	
isig_sem	セマフォ資源の返却
iset_flg	イベント・フラグのセット
ipsnd_dtq	データ・キューへの送信(ポーリング)
ifsnd_dtq	データ・キューへの強制送信

ころですが，一つのシステムに，すべてのサービス・コールを満遍なく使用するということは勧められません．たいがいは後のメンテナンスで困ったことになります．

元々，何でも好き勝手にサービス・コールを使用するとプログラムのサイズが大きくなるという点でも良くありません．プログラムの見通しを良くしたり，理解しやすくするためにも，開発時に以下のような方法を取ることを勧めます．

一つは，いろいろなサービス・コールを好き勝手に使用せず，使用するサービス・コールを限定する「縛り」を設け，システムの作りかたを決めておくことです．

もう一つは，タスク間，設計時のオブジェクト間で通信の方法を決めたり，通信の手段を最初から用意したような大まかなフレームワークのような「共通の通信手段」を用意することです．

筆者は，この二つで何回か窮地を脱したこともあり，ありがたみを感じています．では，このようなありがたみを感じるためには，どのようなサービス・コールを使用すれば良いのでしょうか．メンテナンスを考えるなら，比較的小規模の開発は除いて，特にメール・ボックスの使用を勧めます．これはメッセージを扱っていること，メッセージの中身をコピーしていないことがその理由です．

たとえば，イベント・フラグは少ないビット数でデータをあつかえるうちは良いのですが，仕様がだんだん膨らんだためにビットが足りなくなるという可能性があります．

もう一つの理由として，データのコピーについては，コピーすること自体が処理時間に影響を与えるという点があります．これにはデータ・キューが該当しますが，データが大きければ

コラム

メール・ボックスとデータ・キューに入れるもの

メール・ボックスとデータ・キューは，データをあつかう方法が違います．データ・キューのリング・バッファは入れることのできる上限が決まっていますが，メール・ボックスのリンク・リストには上限がありません．この部分に関してはメール・ボックスのほうがメモリ使用量から見ると有利ではあります．本文中の説明ではコピーする時間だけ不利と書きましたが，どんな条件でも不利なのでしょうか．ここでは，ちょっと違う使いかたについても触れてみたいと思います．

先に，データ・キューは一つあたり，16ビット分のデータを入れるリング・バッファであることを説明しました．実際のアプリケーションでは，この中には何を入れているのでしょうか．バッファ・サイズ0の同期用途には意味のないものも使われますが，一般的には，ちょうど16ビット分のデータとして入れるか，上8ビットのデータを余らせて入れるか，8ビット分のデータとそのときの状態をORするなどして16ビット分を入れるなど，直接データを入れることが多いようです(図A)．

ただ，これ以外のものを入れてはいけないという決まりはありません．そのほかのものを入れることもあります．たとえば，固定長の短冊状の配列データなどを管理するために，配列の要素番号を入れてメッセージの受け渡しに使用する場合もあります．

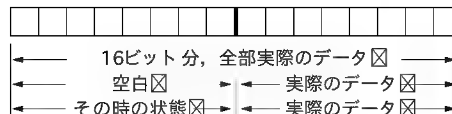
普通ならFIFO状のリング・バッファにそのまま実際のデータを入れると，全部のコピーを作るよりも時間がかかりますが，要素番号で管理できればFIFOに入れるためのコピーは省略できます．共有可能なメモリに配列データを置くことができれば，さらに，データをFIFOバッファから抜き出すときのコピーの処理も省略できます．

ただし，一つのデータのサイズが16ビットという制限があります．ただ，これはshortの範囲内で作れない配列なので，リング・バッファのサイズの上限も大きな問題ではないかもしれません．8ビットchar型では少ないし，32ビットCPUのリニアなアドレスを管理するほど大きな配列は最初から使うこともないとなると，この16ビットは適切なサイズなのかもしれません．

このように書くと良いことずくめのようです．ただ，データ・キューを利用する時点でリング・バッファには即値の実データが入っていると誤解されることも多いため，誤解を招かない範囲での使用を勧めます．誤解を招かないためには，

- ソースにコメントを入れる
- ドキュメントには以下の二つを書く．後でメンテナンスする人にわかるように書くことも必要

1) このシステムではどんな方法で使用しているか，必ず構成と使いかたを書くこと
2) 機能拡張するときのポイント．たとえば，配列要素を増やしたい場合の作業や，もう一つデータ・キューを要しなければならない場合の考慮点など．考えられる範囲は書くこと



図A　よくある16ビットの使い道

リスト 1　メール・ボックスの生成例

```
/* メール・ボックスの生成例　TOPPERSサンプル・ソースより */
CRE_MBX(MBX_ID, {TA_TFIFO, 0, NULL } );
```

大きいほど影響も大きくなります．データをコピーすることで，共有メモリ以外のところでも使用できるなどの利点もあるのですが，筆者はできるだけコピーは少ないほうが良いと考えています．

というわけで，使用するサービス・コールを限定する「縛り」を入れたり，「共通の通信手段」としてサービス・コールを使用する場合には，メール・ボックスを使うことを勧めます．

メール・ボックスの生成

静的API	
CRE_MBX(ID mbxid, { ATR mbxatr, PRI maxmpri, VP mprihd });	
パラメータ	
ID mbxid	メール・ボックスID
ATR mbxatr	メール・ボックスの属性
PRI maxmpri	送信されるメッセージの優先度の最大値
VP mprihd	優先度別のメッセージ・キュー・ヘッダ領域の先頭番地

メール・ボックスの生成は静的API，CRE_MBXを利用します（リスト1）．コンフィギュレータはmboxidを定義し，{}内の値，初期化情報（具体的にはmbxatr，maxmpri，mprihdの値）を利用してメール・ボックスを生成します．

スタンダード・プロファイルの範囲では，mprihd（優先度別のメッセージ・キュー・ヘッダ領域の先頭番地）はサポートしておらず，値にはNULLを入れることになります．

メール・ボックスへの送信

C言語API	
ER snd_mbx(ID mbxid, T_MSG *pk_msg);	
パラメータ	
ID mbxid	送信したいメール・ボックスのID
T_MSG *pk_msg	メッセージへのポインタ
返り値	
ER　E_OK（正常終了）またはエラー・コード	
エラー・コード	
(E_SYS), (E_NOSPT), (E_RSFN), E_CTX, (E_MACV), (E_OACV), (E_NOMEM), E_ID, E_PAR	
E_CTX：コンテキスト・エラー	
E_ID ：不正ID番号	
E_PAR：パラメータ誤り	

メール・ボックスにメッセージを送信します．

タスク外から呼び出した場合，E_CTX（コンテキスト・エラー）が返り，メール・ボックスIDが誤っていた場合はE_ID（不正ID番号エラー）が返ります．E_PAR（パラメータ誤り）は送信されるメッセージの優先度の最大値を超えた場合に発生します．

リスト 2　メール・ボックスの手順と処理をまとめた例

```
/* make_message, setup_messageの戻り値はメッセージへのポインタ */
send_message(Mailbox_ID, make_message(format,
                    "送信メッセージの中身", setup_message()));
```

● メッセージの扱いについて

よく考えるとあたりまえなのですが，以下のことはしてはいけません．

1) 送信後，メッセージを書き換えることはしない．全部書き終わってからsnd_mbxにて送信する．メール・ボックスへ送信するメッセージは，中身を書き終わる前に送信してはいけない．
2) 送信側でメモリ・ブロックを解放することは勧められない．基本的には，メッセージを受け取った側で処理する．思いつきで，受信側でメッセージの処理が済んでいることを確認してからということもできそうだが，何らかの通信が必要になるため，結果的には意味がない．
　共有可能なメモリ上にメッセージを置き，データをそこに書くため，結局は送信後に書き換えることなどができてしまうが，本当に書き換えるとアプリケーションだけでなく，カーネルも正常動作しない．

ここまでのことは，たしかにあたりまえです．しかし，わざと書き換えた事例は聞きませんが，誤って行ってしまったという話は聞いたことがあります．

禁止といっても，機能拡張するうちに追記するデータを書き込むタイミングが送った後になってしまったとか，どこかでポインタがずれて書き換えてしまったとか，本来，発生してはいけないことが発生する場合もあると思います．このような不具合を起こさないためには，どうすれば良いのでしょうか？

まず，機能拡張に関しては，コード設計の時点で，コードをシンプルにし，抽象化しておくのが良いでしょう．たとえば，以下の1)から3)のように分けておけば，よほどの機能追加がない限り，メッセージ・フォーマットの変更などが多数起こっても，変更箇所は迷わないでしょう．

1) メッセージを書くためのメモリ・ブロックの確保，フォーマットに合わせた初期化
2) 実際のデータを書き込みメッセージを作成する
3) メッセージを送信する

手順と処理をまとめるために，リスト2のように一つにしてしまう例もあります．ここまでするとさすがにやりすぎの感はありますが，処理途中に何かが追加されて問題を起こすことは少ないでしょう．

もし，不具合が起こっても，1)から3)それぞれの中で，目的の処理と違うことを行っていないか，3)以降にメモリ・ブロックを書き換えていないことが確認できれば，早い時期に見切りを付けることができます．

あと，どこかでポインタがずれてしまって，たまたまメッセー

ジを書き換えてしまうような不具合に対しては，CPUの処理速度に余裕があるなら2)のところで書き換えられた場合にすぐわかるよう，書いたデータ長を保存しておいたり，空き部分を詰め物で埋めるなどの方法で確認できる仕掛けを入れておくことを勧めます．実際の調査で，いきなりポインタがズレているのでは？などと考えることはないとは思いますが，調査が可能なように作っておくだけでも絞り込みが可能になるので，調査のときには役立ちます．

なお，ここでいう詰め物で埋めるとは，空いているメモリ空間を0xFFなどでフィルすることで，0xFF以外にも判別が付くような特徴あるデータなら何でも使えます．

メール・ボックスからの受信

C言語API	
ER	rcv_mbx(ID mbxid, T_MSG **ppk_msg);
パラメータ	
ID mbxid	受信したいメール・ボックスのID
T_MSG *pk_msg	メッセージへのポインタ
リターン・パラメータ	
ER	E_OK(正常終了)またはエラー・コード
エラー・コード	
(E_SYS), (E_NOSPT), (E_RSFN), E_CTX, (E_MACV), (E_OACV), (E_NOMEM), (E_NOEXS), (E_PAR), (E_DLT), E_ID, E_RLWAI E_RLWAI：待ち状態の強制解除	

メール・ボックスから受信し，メッセージを取り出します．メッセージがない場合は，待ち状態にはいります．

エラーコードに関しては，ディスパッチ保留状態にあるとき，E_CTX(コンテキスト・エラー)を返し，メール・ボックスIDが誤っていた場合はE_ID(不正ID番号エラー)が返ります．

snd_mbxと対になるAPIとして，rcv_mbx，prcv_mbx，trcv_mbxがあります．

先ほどはsnd_mbxで送信の話をしましたが，メッセージをメール・ボックスから取り出した時点で，そのメッセージはカーネルの管理下にありません．破棄するまで使用する側の責任があります．

よくある不具合は，特定の処理の時に破棄する処理を忘れることです．これも，コード設計の時点で，三つのステップに分けておくのがよいでしょう．

1) 受信処理：メッセージの受信処理(メール・ボックスからの受信)．ここでは受け取りと，受け取ったメッセージの正当性を確認するなどの処理
2) データ処理と出力：メッセージの中身から指示された処理を行う
3) メッセージの破棄：後始末

入力，データ処理，出力といった処理の区分とはちょっと違いますが，分けておくことで必ずメッセージの破棄の処理が実施され，メッセージを破棄する箇所がコードの中に埋没していたために破棄する処理を忘れるという不具合を回避できます．

メール・ボックスからの受信(ポーリング)

C言語API	
ER	rcv_mbx(ID mbxid, T_MSG **ppk_msg);
パラメータ	
ID mbxid	受信したいメール・ボックスのID
T_MSG *pk_msg	メッセージへのポインタ
返り値	
ER	E_OK(正常終了)またはエラー・コード
エラー・コード	
(E_SYS), (E_NOSPT), (E_RSFN), E_CTX, (E_MACV), (E_OACV), (E_NOMEM), (E_NOEXS), (E_PAR), (E_DLT), E_ID E_TMOT：ポーリング失敗またはタイム・アウト	

メール・ボックスから受信し，メッセージを取り出します．メッセージがない場合，rcv_mbxでは待ちに入りましたが，prcv_mbxでは待ち状態に入りません．

エラー・コードに関しては，ディスパッチ保留状態にあるとき，E_CTXを返し，メール・ボックスIDが誤っていた場合はE_IDが返ります．メッセージがなかった場合，E_TMOT(ポーリング失敗)が返ります．

メール・ボックスからの受信(タイム・アウトあり)

C言語API	
ER trcv_mbx(ID mbxid, T_MSG **ppk_msg, TMO tmout);	
パラメータ	
ID mbxid	受信したいメール・ボックスのID
T_MSG *pk_msg	メッセージへのポインタ
TMO tmout	タイム・アウト指定
返り値	
ER	E_OK(正常終了)またはエラー・コード
エラー・コード	
(E_SYS), (E_NOSPT), (E_RSFN), E_CTX, (E_MACV), (E_OACV), (E_NOMEM), (E_NOEXS), (E_PAR), (E_DLT),E_ID, E_RLWAI, E_TMOT	

メール・ボックスから受信し，メッセージを取り出します．メッセージがない場合，タイム・アウトの指定時間まで待ち状態に居ます．

エラー・コードに関しては，ディスパッチ保留状態にあるとき，E_CTXを返し，メール・ボックスIDが誤っていた場合はE_IDが返ります．

trcv_mbxは，待ち時間指定によりふるまいを変えることができます．待ち時間の指定を0(ポーリングの指定，TMO_POL)に指定すると待ちに入らず，prcv_msgと同じふるまいになり，逆に待ち時間の指定に無限に指定(カーネル内部では－1を無限時間とみなしており，TMO_FEVRと定義されている)すると，rcv_mbxと同じふるまいになります．

JSPカーネルの1.4以降であれば，各サービス・コールごとにオブジェクト・コードがリンクされるので，rcv_mbx，prcv_mbxと混在しているのであれば，コード・サイズがほんの少し超えてしまった場合に役立つかもしれません．データ・

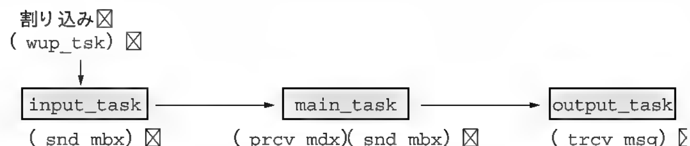


図2　メール・ボックスを用いた処理の流れ

キューでも同じことができるので，覚えておくとよいでしょう．

● 実際にsnd_mbx，rcv_mbxを使ってみる

図2，リスト3に，タスク構成で使った例を示します．以下の例では，四つに別れた構成を取っています．

1) Visul Basicで作成したアプリケーションのボタンを押下するたびに，input_taskを起床させる
2) 起床したinput_taskは，main_tskにメッセージを送信する
3) main_tskでは定期的にメール・ボックスから受信と，動作していることを示す表示を行う
4) output_taskでは一定時間以内にメッセージが到着するかどうかタイム・アウトを見ながら，メッセージの内容，または到着しない旨の表示を行う

このサンプル・コードは，ほかのサービス・コールと同じく，タスクの優先順位によって，メッセージが送信された時点でのタスクの動きが決まります．コンフィギュレーション・ファイルの優先度を変えてみてください．

拡張同期・通信機能

● TOPPERS/JSPでのサポート状況

スタンダード・プロファイルでは拡張同期・通信機能をサポートする必要がないため，TOPPERS/JSPでも拡張同期・通信機能をサポートしていません．そのため，ここではサービス・コールの種別だけをあげておきます．

サポートしていないサービス・コールは，ミューテックス，メッセージ・バッファ，ランデブの三つです．これらの拡張同期・通信機能に該当する機能が必要な場合は，スタンダード・プロファイルではなく，フルセットのカーネル[注]を利用することになります．

おわりに

ここまでで同期・通信機能のサービス・コールを説明し終えたことになります．

リスト3　実際にsnd_mbx，rcv_mbxを使ってみる例

```
void input_task(VP_INT exinf)
{
   ......
   snd_mbx(ID_toMain, ......)
}

void main_task(VP_INT exinf)
{
  while(1){
    ......
    err = prcv_mbx(ID_toMain,......)
    if(err = E_OK){
      snd_msd(ID_toOutput, ......)
    }else if(err = E_TMOT){
      disp("メインタスク処理　システムは動作しています");
    }else {
      disp("内部エラー"):
    }
    dly_tsk(10);
  }
}

void output_task(VP_INT exinf)
{
  while(1){
    ......
    err = trcv_mbx(ID_toMain,......)
    if(err = E_OK){
      disp("正常入力です");
    }else if(err = E_TMOT){
      disp("出力タスク処理　タイム・アウトしました");
    }else {
      disp("内部エラー"):
    }
  }
}
```

お知らせ

TOPPERSカンファレンス

来る6月3日と4日の両日，東京において「TOPPERSカンファレンス」が開催されることになった．TOPPERSの各プロジェクトの報告，会員の発表，そしてInterface誌編集によるパネル・セッション「コア・メンバに聞くTOPPERSプロジェクトの本音」など予定している．

詳細については，TOPPERSのWebサイト(http://www.toppers.jp/)で公開される．　（編集部）

次回は，メモリ・プール管理機能や時間管理機能，時間管理機能の説明に入ります．

きしだ・まさみ　(株)フルノシステムズ

注：フルセットのカーネルも開発が進んでいる．

小型・軽量でSDメモリ・カードと互換性もある

SDIOカード開発入門

第6回（最終回）
SDIOの現況と将来

井手野 雅明

はじめに

本連載第1回（2003年10月号掲載）ではSDIOカードとSDメモリ・カードの違い，そしてSDアソシエーションの紹介およびSDIOを取り巻くマーケットに関して解説した．第2回（2003年11月号掲載）では，SDIOカードを設計するうえで非常に重要なSDIO標準規格の概要について触れた．第3回（2004年1月号掲載）では，そのSDIO標準規格に基づき開発されたIEEE802.11b SDIO無線LANカード（写真1）を例に無線LAN通信とSDIOのつながりについて解説し，第4回（2004年2月号掲載）および第5回（2004年4月号掲載）では，実際にSDIOカードのプロトタイプ設計事例をシイガイズ社製SDIO開発ボード（写真2）を使って解説した．

連載の最終回となる今回は，SDIOカード市場近況から，SDIOカード市場のトレンド，そしてサービスを含めたSDIO市場全体の将来像について述べたいと思う．

一般的にSDIOはいまだ広く普及してはいないが，多くの人が最近SDIOという名前を耳にしたり，見たりする機会が増え，“SDIOとはなんだろう？”と興味をもつ方が確実に増えていると思われる．本連載はそういう方に向けて執筆したガイドであり，今までの連載を読んでもらえればSDIOに関しての基本的な知識や市場動向をつかんでもらえると思う．SDIOカード市場の拡大は，これからどれだけ多くのSDIOカードが登場するかにかかっている．ぜひとも本連載をきっかけとして，より多くのアイデアに富んだSDIOカードが市場に登場することを期待している．

おさらい「SDIOとは？」

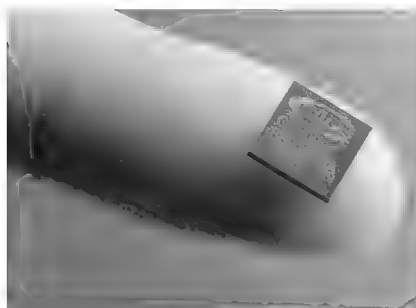
SDIO（エス・ディー・アイ・オー）とは，Secure Digital Input/Outputの略称であり，SDアソシエーションで策定されたSDメモリ・カードと同様の標準規格の一つである．一般的によく知られるSDメモリ・カードとはその外形（長さ32mm×幅24mm×厚さ2.1mm）の長さ方向に関する規格が異なり，横幅ならびに厚さに関してはSDメモリ・カードとまったく同じである．SDメモリと同一形状を長さ37mmまで保てば，それ以外の形状（長さ，厚さ，幅）は基本的に自由である．端子数もSDメモリ・カードと同じ9端子で，ピン配置位置も変わらず電気的条件も同等である（写真3）．

SDIOカード市場近況

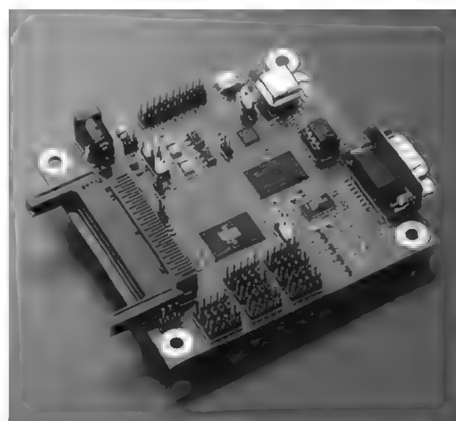
2004年3月現在，すでに発売が開始されているSDIOカード



写真1　IEEE802.11b SDIO無線LANカード（SD-Link11b）



（a）CG100



（b）CG100EDK

写真2　SDIOコントローラCG100（左）とSDIO開発ボードCG100EDK（右）

は，IEEE802.11b無線LANカード，Bluetoothカード，GPSカード，31万画素CMOSディジタル・カメラ・カード，バーコード・スキャナ・カード，FMラジオ・カード，PHSカードなどである．今後はさらに，ディジタル・テレビ・チューナ・カード，IEEE802.11g無線LANカード，Bluetooth Ver.2.0カード，高画素ディジタル・カメラ・カードなど非常に多くの製品企画が進んでいる（図1）．

筆者の実感として，ここ半年でアジア圏の企業によるSDIOカード開発が非常に活発化してきている．また，北米企業によるSDIOカード業界への新規参入が多く見られ，特にコンピュータ周辺機器の企業がPCカード，CFカードの次に主流になるカードという見方で対応を加速しているように見える．一方，SD/SDIOカード・スロットを積極的に採用しているPDAも，最近は低価格化への傾向が強く，およそ2～4万円の価格帯で十分高性能なPDAを購入できるようになった．SDIOカード市場の近未来（2004年～2005年）では，昨年から今年にかけて発売が開始されたSDIOカード＋低価格PDAでの各種サービス事業が本格化するのではないかと予想される．その中でもっともビジネスの中心として注目されているのが，SDIO無線カード（IEEE802.11b，Bluetooth，PHSなど）を応用したビジネス・モデルである．

SDIOカードで加速するPDAのネットワーク化

現実的にSDIOはすでにPDA（Personal Digital Assistance）においてはほぼ，市民権を得たといっても過言ではない．昨今は登場する新製品のほとんどがCF（Compact Flash）カードのインターフェースではなく，SD/SDIOを標準カード・インターフェースとして採用している（図2）．PDAのSD/SDIO対応機種が増加したのは，SDIOカードのサイズが小さく，また薄く



写真3　SDメモリ・カード（左）とSDIOカード（右）の外観（裏面）

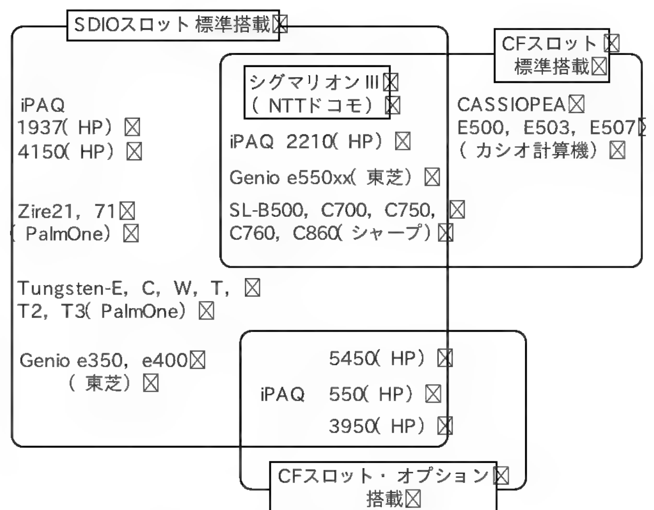


図2　SD/SDIOを採用したおもなPDA

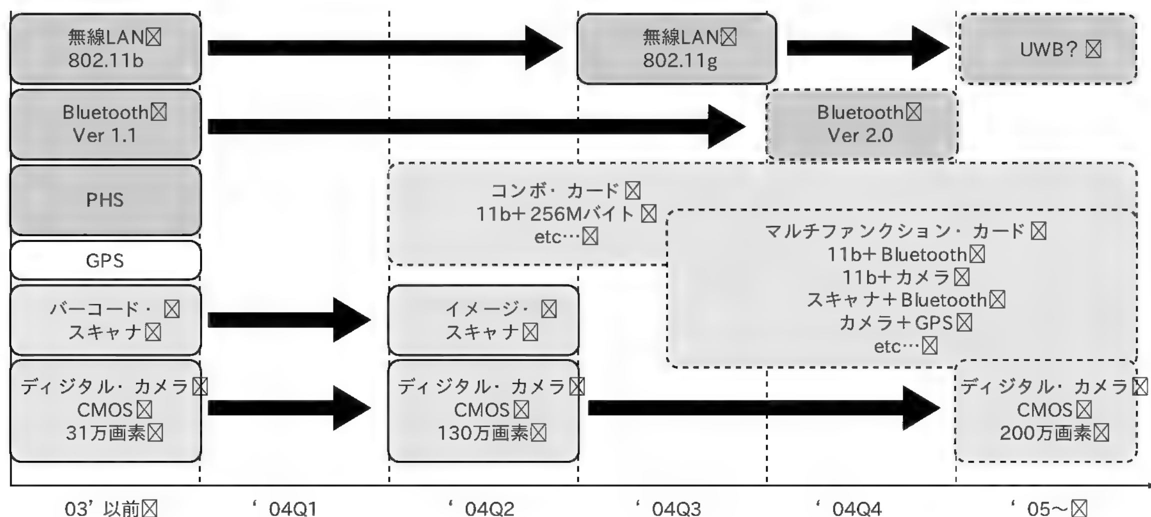


図1　SDIOカードのロード・マップ

軽いことがPDAのモバイル性(小型・軽量化)を重要視する市場要求に合致しているということ，さらにメモリ・カードとI/Oカードの両面のソリューションをもっているということが大きな理由であるといえる．

さらにPDAのネットワーク化に拍車をかけたのが，PocketPC2003の登場である．従来PCとPDAの間のデータ・シンクロ(同期)は，PCからUSBでクレードルに接続して有線で行われてきたが，これが無線ネットワークでもシンクロができるようになった．設定しておけば，アクセス・ポイント(またはPHS基地局など)との通信ができる環境下で，随時PCとのシンクロが可能となり，情報がリアルタイムで更新されるようになるばかりでなく，PC内のデータ(またはサーバ)に格納されたデータを参照し，活用できるようになった．従来，無線LANモジュールを内蔵した高級機タイプのPDAでしか利用できなかったこれら機能が，SDIO無線LANカードの登場で低価格のPDAでも利用できるようになった．

最近ではホーム・サーバに録画したテレビ番組を，遠隔地からインターネット経由でダウンロードし(またはSDメモリに格納)，PDAなどで楽しむことができる製品やハードディスク付きDVDプレーヤも発売されている．したがって，PDAはすでにスタンドアロンで使われるものではなく，ネットワークの中で稼動する端末であるという位置付けになりつつあることにまちがいない．

もとよりPDAは個人情報を管理する手帳的な使用用途に限られてきたが，これからはそのモバイル性を有効的に活用できる動画ビューワであったり，情報を発信する端末であったりと，その発展性はSDIOカードを介した無線ネットワークとのつながりにより大きな可能性を持ち始めている．

いつもの駅が情報ステーション“無線LAN倶楽部”

昨年より非常に興味深いサービスが提供されているのをご存じだろうか？　それはNTTブロードバンドプラットホーム(株)が，首都圏の主要な駅で提供している“無線LAN倶楽部”(http://www.c-guys.net/wlanclub/wlan_index.html/)というサービスである．

駅のホームなどに利用可能であることを示す目印(写真4)が貼ってあり，その近辺であれば無線LANを使用することができる．これはあらかじめWeb登録(有料)を行い，入手したIDなどをPDAに設定するだけで使用できる．

おもしろいのは，従来のWebページのブラウジングやEメールが利用できるだけでなく，PDA向けのコンテンツ・デリバリ・サービスである“コンテンツシンクロ”を提供している点である．コンテンツは各種用意されており，新聞，雑誌そのほか多彩な情報をホームでまとめてダウンロードし，電車の中でチェックできる．これは，この忙しい世の中とても便利なことではないだろうか？

最近では駅だけではなく，コーヒー・ショップや空港などでも利用できるようになってきている．もちろん無線LANを出入り口としてPDAにSDIOカードを挿入し，GPS用地図データのダウンロードなど，SDIOカードをさらに効果的に活用できるサービスも駅を中心に提供されはじめている．



写真4
無線LAN倶楽部ステッカ

市民運動で無線LANネットワーク拡大を推進

PHSや携帯電話の場合は基地局がかなり高価であるため，大手のキャリアが計画的に基地局を配置し，サービスを展開してきた．それゆえ，サービスを拡大させるスピードにもおのずと限界があり，設備投資が大きいので通信コストもそれなりにかかる．しかし，無線LANの場合はカバーエリアが狭いとはいえ，アクセス・ポイント(基地局)の初期コストは1～2万円，月額数千円のランニング・コストでまかなえる．それも企業ではなく個人ベースで広域ネットワークの一翼を担うことができる．そのような壮大な計画を実践しているのがNPO(非営利)団体の“みあこネット”である(http://www.cguys.net/partner/miakonet.html/)．

店舗や個人などがそれぞれ，みあこネットのVPN環境にアクセス・ポイントを無償で提供し，登録を行ったユーザは無償で使用できるサービスを行っている．現在，おもに京都市内で無線LAN環境を着々と整備しており，特に視覚障害者が観光地を訪れた際に観光ガイドをダウンロードし，自動で読み上げてくれる“ユビキタス・ラジオ”(写真5)などがすでに稼動している．

またPDA＋SDIOカメラ＋無線LANを使って，ボランティアが視覚障害者の目の代わりとなり，音声で障害者を遠隔サポートするという実験も計画されている．対象地域も本拠地の京都を飛び出し，日本各地に拡大中であり，これからが非常に楽しみである．

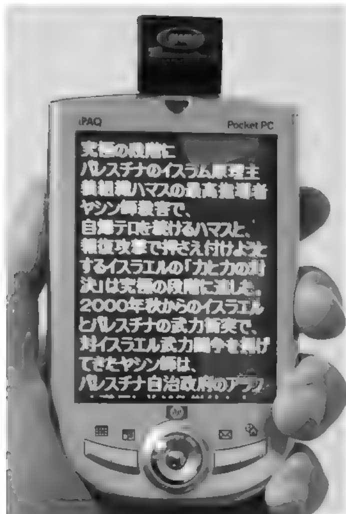


写真5　ユビキタス・ラジオの操作画面



写真6　携帯IP電話

SDIOスロット搭載の携帯電話が登場!?

アクセス・ポイントが一般化し，街中で無線LANが使える環境が整備されると，登場してくるのがVoIPであり，携帯IP電話である．これは新世代携帯電話として期待されているが，基本的に電話とは"通話中に切れない"ということと，"どこでもつながる"ということが重要であり，その点で現状の通信事業形態からすぐにIP電話事業に変わることは難しいといわれている．IP電話サービスはキャリア企業の収益を左右する大きな事がらなので，実現には相当時間がかかるともいわれているが，ブラウザとしての無線LANの広帯域伝送は魅力的だろう．近い将来，携帯電話器にSDIOスロットが搭載され，SDIO無線LANカードを介しての広帯域伝送が実現されるかもしれない．

SDIOカードを搭載した携帯IP電話の登場

昨年秋にアイピートーク(株)(http://www.iptalk.net/index.htm)から，SDIOカードを標準搭載した携帯IP電話が発表された(写真6)．アイピートーク社はプロバイダを問わず，またブロードバンドの種類を問わず，現在稼動中の企業内交換機による電話機の電話番号もそのままで，広域無料内線網と格安外線電話が実現する画期的なサービスとして設計されている．現在稼動中のビジネスホン主装置の外線側にIPTalk-Pro端末を接続して実現することにより，通信コストを大幅に削減できる．

携帯IP電話は今のところ，上記システムの一部として構内電話としての利用目的が主であるが，将来もし街に飛び出すことがあれば，非常に楽しみなソリューションになるだろう．ちなみに，この携帯IP電話はSDIOスロットを背面にもち，SDIO無線LANカード"SD-Link11b"(シイガイズ社製)が標準搭載されている．

携帯電話とPDAの融合"スマートホン"

すでに欧州や韓国などで広がりを見せているスマートホン(PDA＋携帯電話)では，普段はGSMやCDMAで携帯電話として利用し，アクセス・ポイントがある環境ではSDIO無線LANカードを使用することにより無線LAN機能付PDAとして利用できる．

ここでよくある質問として，PDAに無線LANモジュールがいずれ標準搭載されれば，SDIO無線LANカードはほとんど利用されないのではないかという点である．まずすべてのPDAユーザが無線LAN通信を必要としているわけではないので，標準搭載することは市場要求と要求コストを比較検討すると，今はまだ見合わないという声が多い．また，モジュールを組み込むことは開発上の負荷でもあり，いろいろなリスクをともなう．とりわけ，無線認証という法律の壁があり，さらに海外輸出となると機器個別に認証を取らなければならないので，その労力と費用が莫大である．

SDIOカードを採用した場合，カードですでに無線認証を得ているのであれば，これら無線認定獲得にかかわる作業はほとんど必要がなくなり，無線LANモジュール内蔵製品に比べ，早期に製品を市場に投入できる．本格的に無線LANモジュールが標準搭載されるまではこのソリューションが主流になるのではないかと思われる．もちろんそれ以外のSDIOカード，たとえばSDIO BluetoothカードやSDIO GPSカードなども使用できる．

これらスマートホンは，日本でもすでに国産メーカがDoPaやFOMAなどが使える機種を業務用として提供し実際に利用

されている．一般ユーザ向けに発売を開始する予定があるかもしれないが，PDA そのものを持って電話をする格好が，超小型の携帯電話に慣れてしまった日本人にとっては抵抗感があるかもしれない．

SD アソシエーションの最近の活動状況

今まで述べたいろいろなビジネス形態や SDIO 関連機器開発も，SD アソシエーションという標準団体の活動なくして語ることはできない．連載第 1 回で説明したように，SD アソシエーションは 1999 年 8 月に松下電器産業(株)，(株)東芝，SanDisk Corporation 社の 3 社で設立された，おもに SD メモリ・カードの共同開発，業界標準化およびプロモーションを目的とした団体である．SD アソシエーションは昨年までに Mini-SD メモリ規格などを世に提供し，これからも多くのメンバ間で議論を重ねて新しい規格作りに積極的に取り組んでいく．エグゼクティブ・メンバになれば実際の規格決定の投票権を得られるので，まさに自分の意見を直接反映させることができる民主的な団体である．

SD アソシエーション東京オフィスの岩本レプレゼンタティブによれば，今年 2 月にハワイで開催された SD アソシエーション総会(毎年 2～3 回開催)では，新メンバなどに向けてのセミナなどが開催され，今まで SD/SDIO になじみのない方でもその概要を理解していただくことができたようである．また Mini-SD や Mini-SDIO，さらには次世代の SD カードに関するディスカッションも活発に行われた．もちろん，参加者は SD/SDIO カードだけではなく，PC 周辺機器関連企業やディジタル家電大手企業など，ホスト機器を開発するメンバも多数参加した．今後の活動としては，今年の秋ごろに総会を開催する予定で，さらに 2004 年 10 月 20 日～23 日に東京ビッグサイトで開催される WPC エキスポ 2004 に，SD アソシエーションとして出展を行う予定である．この SD アソシエーション・ブースにはメンバであれば出展することができ(場合により，有料，抽選などあり)，マーケティング・コミッティに参加し，展示に関して，自分のアイデアを提案し，議論を行うこともできる．昨年は CEATEC2003 で SDIO 専門のコーナを設け，タッチ&プレイ・デモを行った．これは SDIO が世に認知されるにはたいへん良い機会になったことはいうまでもない．そのほかのメンバ展示ブースでは，いろいろな SDIO 機器やカードが展示され，活発な商談が展開されていた．詳しくは SD アソシエーションの Web サイト(`http://www.sdcard.org/`)か，連載第 1 回にある SD アソシエーションに関する記事を参照してもらいたい．

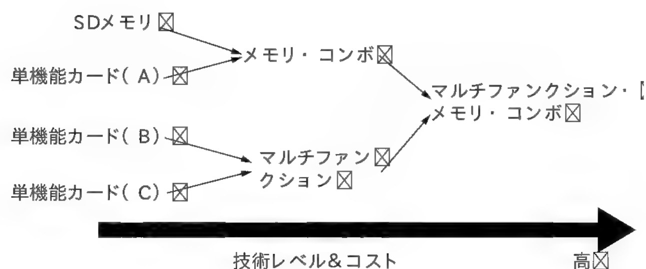


図 3　SDIO カードの将来のトレンド

SDIO カードの将来展望

中長期の SDIO カードのトレンドは，大きく三つの流れに分かれていくと推測される．一つは単機能追求で最先端アプリケーションを Time-To-Market で市場投入し先行者利益を得る単機能型 SDIO カード，二つ目は単機能カードに SD メモリを搭載するコンボ型 SDIO カード，三つ目は単機能カードどうしを結合し，複合型とするマルチファンクション型 SDIO カードである(図 3)．

● 先端機能実現に特化したカード "単機能型 SDIO カード"

まず最初に挙げられるのが IEEE802.11a/g だろう．市場的にはすでに IEEE802.11b が一般化し，11a/b/g の複合型無線 LAN が主流となりつつある．IEEE802.11a/b/g の SDIO 無線 LAN カードが登場することにより，さらに多くのビジネス・チャンスが登場してくると思われる．やはりこれは有効伝送能力が向上したことが大きく，IEEE802.11b ではビデオ・レートを扱うには少々厳しい．11a/g であれば実質的に見ても MPEG-2 データを伝送することが可能なので，一気にディジタル家電やディジタル AV 機器への展開が広がるだろう．

それ以外の単機能型の SDIO カードとしては，IC タグを認識する SDIO カード，指紋認識セキュリティ SDIO カード，何らかの特殊なセンサを内蔵した SDIO カードなど，いろいろなアプリケーションが考えられる．

ユーザとしては機能がカードとして独立しているので，欲しい機能のカードを購入でき，ホスト機器製造側では内蔵しなくともカードにより市場要求を満足できるので，リスクの回避につながるといったメリットがある．この単機能型 SDIO カードが一番基本的な SDIO カードのビジネス・アプローチではないかと思う．

● 単機能+メモリ混載カード "コンボ型 SDIO カード"

次に単機能 SDIO にフラッシュ・メモリを搭載する SDIO メモリ・コンボ・カード型であるが，実際に SanDisk では IEEE802.11b の SDIO カードに 256M バイトのフラッシュ・メモリを内蔵した SDIO メモリ・コンボ・カードの発売を開始すると発表した．コンボ型カードのメリットは，機器本体の記憶容量が少ない場合や，SDIO スロットが一つしかなく SDIO カードと SD メモリ・カードを同時に使用できない場合，また，ディジタル・カメラ・カードで撮影した画像データの格納，GPS カードでは地図データの格納など，応用範囲は意外と広い．一方，デメリットとしては，メモリが付加される分，製品単価が

上昇する，ある程度メモリ容量が決められてしまう，64Mバイト用，128Mバイト用，256Mバイト用…など，搭載メモリ容量ごとにラインナップをそろえることは販売戦略上効率的ではない，メモリ・カードのように手軽に人に渡したりできず，渡してしまったら，そのカードが返ってくるまでその機能が使えなくなってしまうといった点がある．

実際にSDIOメモリ・コンボ・カードを開発するときには，まずはSDアソシエーションに問い合せ願いたい．肝心なのは，混載するアプリケーションに最適な組み合わせのメモリ容量を見極めることである．

● 機能複合型カード“マルチファンクション型SDIOカード”

現時点でマルチファンクション型SDIOカードに関する発表は，今のところ筆者の知る限り見当たらないが，IEEE802.11bとBluetoothの複合型カードが登場するのではないかという予想がある．同じ2.4GHz周波数帯域を使用することなどから，部品の共有化が計れるからではないかというのが根拠のようだ．

これ以外では，IEEE802.11bとGPSカードをマルチ化し，特定のアクセス・ポイントに接近し，接続した段階でマップ・データやそれ以外の情報などをダウンロードさせるといったことなどが検討されている．さらにマルチファンクション型SDIOカードにメモリを搭載して，“マルチファンクション＆コンボ型SDIOカード“というユニークなSDIOに発展する可能性もある．

いずれにしろ，サイズ的に部品の積載能力におのずと限界があるので，あれもこれもというわけにはいかない．むしろこれからはホスト側機器にもたせる機能，カードで処理すべき機能を，SDアソシエーションの中でホスト機器開発企業とSDIOカード開発企業が議論し，効率的な分担を取り決めていくことが重要であると思う．

まずはSDアソシエーションに加入しよう！

SDIOカードを開発しようと心に決めたら，まずはSDアソシエーションに加入することが先決である．加入するにはWebサイト(http://www.sdcard.org/)の“How to Join SDA”を選択し，そこに記載されている手続きを行うことで後日，加入案内が送付され，メンバとして承認される．

メンバは，ジェネラル(General)メンバとエグゼクティブ(Executive)メンバの2通りがある．ジェネラル・メンバは会議参加とメンバに開示される情報にアクセスする権利を得ることができる．エグゼクティブ・メンバは会議での決定に対する投票権をもち，メンバに開示される情報にアクセスする権利を得ることができるだけでなく，SDアソシエーション役員に立候補することができる(所定の条件を満たすことが必要)．本格的にSDアソシエーションで活動を行いたいのであれば，エグゼクティブ・メンバになる必要がある．

製品を開発し，発売を行うためには別途HALA(Host/Ancillary Product License Agreement)のライセンスを受ける必要がある．これによりSDに関する一部特許の使用が許可され，SD，MiniSD，SDIOのロゴを入手し使用する権利を得ることができる．詳しくは，http://www.sdcard.org/を参照してほしい．

まとめ

以上，約半年間に渡ってSDIOに関する概要説明を行ってきた．この連載がこれからSDIOカードを開発しようという方のガイドとなれば幸いである．

詳しい情報はSDアソシエーションに加入しないと入手できないので，先に述べたように，まずは加入して資料を入手し，会議にも積極的に参加して，SDアソシエーションを盛り立ててほしいと思う．

SDIOはこれからのモバイル機器に対して非常に有用なインターフェースとなる規格であり，今から市場参入しておくことは決してむだにはならないと確信している．

いでの・まさあき　シイガイズ(株)マーケティングディレクタ

やり直しのための信号数学 第24回

総まとめⅠ（直交変換編）



三谷 政昭

ほぼ3年間という長丁場であった「信号数学」の連載では，信号解析の土台をなす話題として“直交変換”，“DFT（ディジタル・フーリエ変換）”，“FFT（高速フーリエ変換）”，“DCT（ディジタル・コサイン変換）”について説明してきた．いずれの信号変換も重要であり，数学的な取り扱いに習熟しておくことが信号処理技術者の基礎知識として必須であることに疑問の余地はないと確信する．

本連載を終えるにあたり，今回からは「信号数学の総まとめ」として，みなさんといっしょに学んできた信号数学のエキスをもう一度しっかりと理解し，身に付けてもらうことを目標に学習を進めていくことにする．

今回は「直交変換編」と題し，“正規直交基底ベクトルによる信号表現”を用いて，信号相関，DFT，DCT，パーシバルの定理などについて具体的な数値例を示しながら，わかりやすく解説する．（筆者）

信号変換の利点

信号の変換方法には多種多様な種類があり，それぞれ固有の特徴をいかす形で，いろいろな分野に応用されている．ここでは，直交変換としてよく知られているDFTとDCTを採り上げ，信号を変換することの必要性と，その利点について説明する．

● 信号の性質を調べる

図24.1（a）の16サンプルのディジタル信号$\left\{x_k = x(kT)\right\}_{k=0}^{k=15}$をDFTすると，図24.1（b）が得られる．ここで，16個のDFT値$\left\{X_\ell = X(\ell\Delta f)\right\}_{\ell=0}^{\ell=15}$のうち，4個以外はゼロ（0）であり，しかも独立な値はわずか二つしかない．このことから，ディジタル信号はかなり複雑な形状をしているが，実は二つのcos波を合成したものであることがわかる．つまり，DFT，すなわちフーリエ変換は信号をcos波（あるいはsin波）に分解する働きを有しているのである．

このようにDFTを用いて信号を周波数ごとに分解することで，信号に含まれるcos波の大きさ，周波数および位相の三つのパラメータを同時に知ることができる．ただ，DFT以外の変換方法は，一般にcos波成分への分解には対応しないが，信号の分解法の一つに位置づけられる．

ところで，信号分解による違いは，分解する際の変換基底に

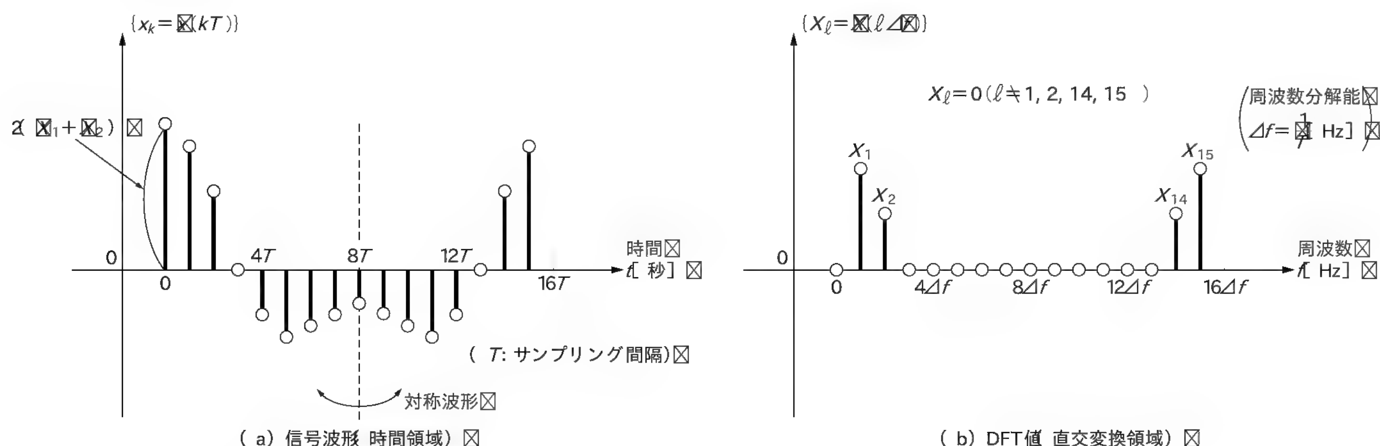


図24.1　ディジタル信号$\left\{x_k\right\}_{k=0}^{k=15}$のDFT値$\left\{X_\ell\right\}_{\ell=0}^{\ell=15}$



応用分野	主たる内容
線形システム	線形システムの出力のフーリエ変換は，システムの伝達関数と入力信号のフーリエ変換の積で与えられる
アンテナ工学	アンテナの放射パターンはアンテナ開口面における電流分布のフーリエ変換で与えられる
光学系システム	集光レンズの前面と背面の焦点面での光の振幅分布がフーリエ変換の関係にある
不規則過程	不規則過程の電力スペクトルは，その過程の自己相関関数のフーリエ変換で与えられる
確率論	不規則(確率)変数の特性関数は，その変数の確率密度関数のフーリエ変換として与えられる
量子物理学	量子論における不確定性原理は基本的にはフーリエ変換に関係している．たとえば，粒子の運動量と存在する位置とは互いにフーリエ変換の関係にある
境界値問題	偏微分方程式の解はフーリエ変換を用いて算出することもできる

図 24.2　フーリエ変換(DFT，DCT)に基づく信号分解による応用例

依存しており，「どのような信号を調べようとしているのか」，また「どのような性質を知ろうとしているのか」により，適用すべき信号変換法は異なってくる．

とにかく，信号変換することによって，時間波形の形状を見るだけではわかりにくい信号の特徴や性質が鮮明な形であぶり出されてくるのである(図 24.2)．

なお，参考までに図 24.1(a)の信号を DCT した値を図 24.3 に示しておく．図 24.1(b)の DFT 値と見比べるとわかるように，「同じ時間波形であっても，異なる変換法に対しては，異なる変換値が得られる」ことになるのである．

● 信号を処理する

信号分解によって得られる変換値を利用して，雑音を除去したりとか，輪郭を強調するといった種々の信号処理を少ない演算処理でもって効率的に実行することができるようになる．

図 24.1(a)の時間波形から一つの cos 波のみを抽出してみよう．処理手順は，図 24.1(b)の DFT 値から一つの周波数成分を選び出して，図 24.4(a)の DFT 値を考えればよい．これを IDFT(ディジタル逆フーリエ変換)して，時間信号に戻せば図 24.4(b)の結果が得られ，当然のことながら選ばれた周波数の cos 波となる．

● 信号のもつ相関を変える

まず，図 24.1(a)は 16個の信号値を用いて時間波形を表しているが，同図(b)ではわずか 4個の変換値で同じ信号の表現が可能であることを示すことになる．

つまり，cos 波の三つのパラメータ(信号の大きさ，周波数，位相)を，時間信号では 16個の信号値の相互関係として持っていることを意味している．これに対して，変換値は直接 cos 波の三つのパラメータを時間信号とは独立な情報としてもつことになる．もっと言えば，時間信号は 16個の各信号値間に相関があるわけで，むだな情報が含まれ，変換値は無相関(相関がない)で必要最小限の情報しかないのである．

したがって，信号を送受信したり，メモリに記憶させること

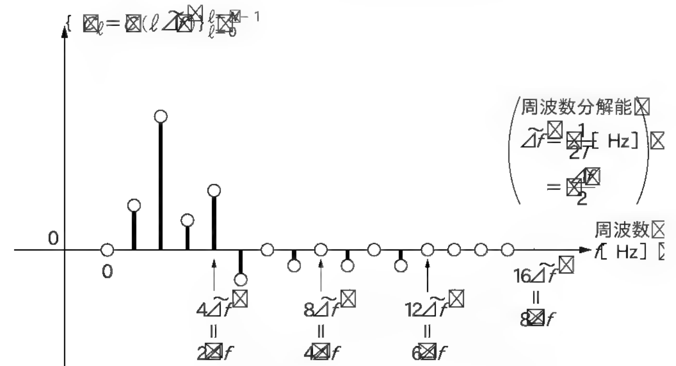
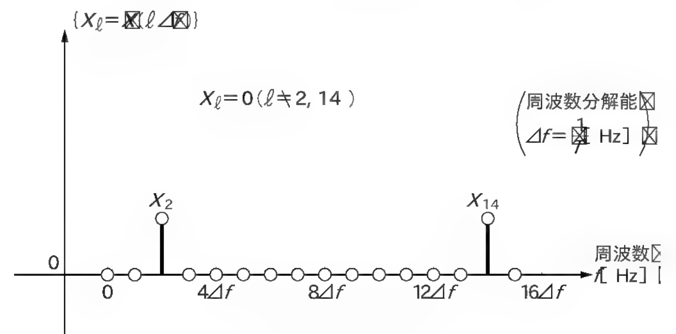
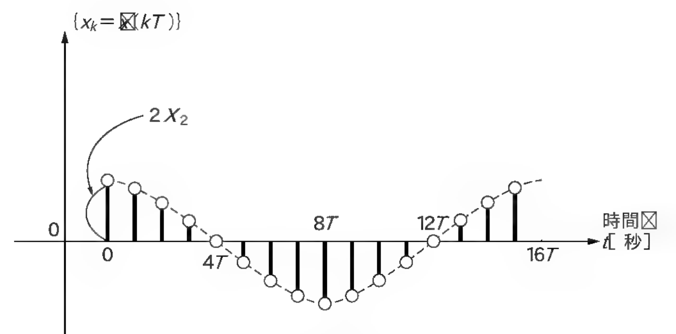


図 24.3　図 1(a)のディジタル信号 $\{x_k\}_{k=0}^{k=15}$ の DCT 値 $\{C_\ell\}_{\ell=0}^{\ell=15}$



(a) 選択抽出したDFT値



(b)(a)のIDFT値

図 24.4　DFT 値(X_2と X_{14})の選択抽出

を考えた場合には，明らかに変換値を用いて無相関な情報に置き換えたほうが少ないデータ量で済む．このように信号に適切な変換を適用することにより，信号のもつ情報を表現するために必要なデータ量を削減することができる．情報のむだな部分を取り除くことに等価な処理であり，データ圧縮(むだな情報を取り除いて，必要最小限のデータ量にすること)を可能にする．

正規直交基底と信号表現

次に，信号変換の利点を理解してもらえたことを受けて，1次元データの正規直交基底による信号表現で必要となる“基底ベクトル”について説明しておこう．

いま，1次元データ(N個)に対する基底ベクトルを，

$$\boldsymbol{\phi}^{(\ell)}=\left[\phi_0{}^{(\ell)},\phi_1{}^{(\ell)},\phi_2{}^{(\ell)},\cdots,\phi_{N-1}{}^{(\ell)}\right]\quad;\ell=0,1,2,\cdots,N-1 \quad\cdots\cdots(1)$$

とし，また二つの基底ベクトル$(\boldsymbol{\phi}^{(k)},\boldsymbol{\phi}^{(\ell)})$の内積を，

$$\left\langle\boldsymbol{\phi}^{(k)},\boldsymbol{\phi}^{(\ell)}\right\rangle=\frac{1}{N}\sum_{n=0}^{N-1}\phi_n{}^{(k)}\overline{\phi_n{}^{(\ell)}} \quad\cdots\cdots(2)$$

ただし，$\overline{\phi_n^{(\ell)}}$ は $\phi_n^{(\ell)}$ の複素共役で定義する．ここで，

$$\left\langle\boldsymbol{\phi}^{(k)},\boldsymbol{\phi}^{(\ell)}\right\rangle=\begin{cases}1 & ;k=\ell\\0 & ;k\neq\ell\end{cases} \quad\cdots\cdots(3)$$

の条件を満たすとき，「$\left\{\boldsymbol{\phi}^{(\ell)}\right\}_{\ell=0}^{\ell=N-1}$は“正規直交基底ベクトル”である」という．なお，式(3)で基底ベクトルの内積が‘1’になることが“正規性”，また‘0’になることが“直交性”という性質を意味する．

いま，図24.5に示すN個のサンプル値からなるディジタル信号sを，

$$s=\left\{s_n\right\}_{n=0}^{n=N-1}\quad;\ s_n=s(nT) \quad\cdots\cdots(4)$$

と表すとき，正規直交基底ベクトル$\left\{\boldsymbol{\phi}^{(\ell)}\right\}_{\ell=0}^{\ell=N-1}$を用いれば，

$$s=G_0\,\boldsymbol{\phi}^{(0)}+G_1\,\boldsymbol{\phi}^{(1)}+G_2\,\boldsymbol{\phi}^{(2)}+\cdots+G_{N-1}\,\boldsymbol{\phi}^{(N-1)}$$
$$=\sum_{\ell=0}^{N-1}G_\ell\boldsymbol{\phi}^{(\ell)} \quad\cdots\cdots(5)$$

となり，正規直交表示できる．なお，式(5)に現れる変数$\left\{G_\ell\right\}_{\ell=0}^{\ell=N-1}$は，“展開係数”と呼ばれる．

また，式(5)よりディジタル信号値$\left\{s_n\right\}_{n=0}^{n=N-1}$は，式(1)の要素値を用いて，

$$s_n=G_0\,\phi_n{}^{(0)}+G_1\phi_n{}^{(1)}+G_2\phi_n{}^{(2)}+\cdots+G_{N-1}\,\phi_n{}^{(N-1)}$$
$$=\sum_{\ell=0}^{N-1}G_\ell\phi_n{}^{(\ell)} \quad\cdots\cdots(6)$$

と表される．

それでは，展開係数G_ℓを求める計算手順を示す．まず，式(3)の条件を考慮して，ディジタル信号ベクトルsと正規直交基底ベクトル$\boldsymbol{\phi}^{(\ell)}$との内積を計算すると，

$$\left\langle s,\boldsymbol{\phi}^{(\ell)}\right\rangle=G_0\underbrace{\left\langle\boldsymbol{\phi}^{(0)},\boldsymbol{\phi}^{(\ell)}\right\rangle}_{0}+G_1\underbrace{\left\langle\boldsymbol{\phi}^{(1)},\boldsymbol{\phi}^{(\ell)}\right\rangle}_{0}+\cdots$$
$$+G_\ell\underbrace{\left\langle\boldsymbol{\phi}^{(\ell)},\boldsymbol{\phi}^{(\ell)}\right\rangle}_{1}+\cdots+G_{N-1}\underbrace{\left\langle\boldsymbol{\phi}^{(N-1)},\boldsymbol{\phi}^{(\ell)}\right\rangle}_{0}$$

であり，展開係数G_ℓは，

$$G_\ell=\left\langle s,\boldsymbol{\phi}^{(\ell)}\right\rangle=\frac{1}{N}\sum_{n=0}^{N-1}s_n\overline{\phi_n{}^{(\ell)}} \quad\cdots\cdots(7)$$

と表される．このとき，式(7)の総和(Σ)計算に際しては，正規直交基底ベクトルの各要素$\left\{\phi_n{}^{(\ell)}\right\}_{n=0}^{n=N-1}$の複素共役$\left\{\overline{\phi_n{}^{(\ell)}}\right\}_{n=0}^{n=N-1}$と信号値$\left\{s_n\right\}_{n=0}^{n=N-1}$との積和になることをしっかりと記憶に留めておいてもらいたい．

例題1

いま，サンプル数$N=3$に対する正規直交基底ベクトル$\left\{\boldsymbol{\phi}^{(\ell)}\right\}_{\ell=0}^{\ell=2}$を，

$$\begin{cases}\boldsymbol{\phi}^{(0)}=\begin{bmatrix}1 & 1 & 1\end{bmatrix}\\ \boldsymbol{\phi}^{(1)}=\left[\sqrt{2}\cos\left(\frac{\pi}{6}\right)\ \sqrt{2}\cos\left(\frac{3\pi}{6}\right)\ \sqrt{2}\cos\left(\frac{5\pi}{6}\right)\right]\\ \boldsymbol{\phi}^{(2)}=\left[\sqrt{2}\cos\left(\frac{2\pi}{6}\right)\ \sqrt{2}\cos\left(\frac{6\pi}{6}\right)\ \sqrt{2}\cos\left(\frac{10\pi}{6}\right)\right]\end{cases} \quad\cdots\cdots(8)$$

とするとき，正規直交性を有することを検証せよ．また，展開係数$\left\{G_\ell\right\}_{\ell=0}^{\ell=2}$と信号値$\left\{s_n\right\}_{n=0}^{n=2}$の相互関係式を示せ．

解答1

● 正規直交性

式(3)が成立することを示せばよい．2～3の典型的な計算例を示しておくので，省略したものについては計算例を参考に検証してもらいたい．

$$\left\langle\boldsymbol{\phi}^{(1)},\boldsymbol{\phi}^{(1)}\right\rangle=\frac{1}{3}\left\{\sqrt{2}\cos\left(\frac{\pi}{6}\right)\times\sqrt{2}\cos\left(\frac{\pi}{6}\right)\right.$$
$$\left.\sqrt{2}\cos\left(\frac{3\pi}{6}\right)\times\sqrt{2}\cos\left(\frac{3\pi}{6}\right)+\sqrt{2}\cos\left(\frac{5\pi}{6}\right)\times\sqrt{2}\cos\left(\frac{5\pi}{6}\right)\right\}$$
$$=\frac{2}{3}\left\{\cos^2\left(\frac{\pi}{6}\right)+\cos^2\left(\frac{3\pi}{6}\right)+\cos^2\left(\frac{5\pi}{6}\right)\right\}$$
$$=\frac{2}{3}\left\{\left(\frac{\sqrt{3}}{2}\right)^2+(0)^2+\left(-\frac{\sqrt{3}}{2}\right)^2\right\}=\frac{2}{3}\cdot\frac{3}{2}=1$$

$$\left\langle\boldsymbol{\phi}^{(0)},\boldsymbol{\phi}^{(1)}\right\rangle=\frac{1}{3}\left\{1\times\sqrt{2}\cos\left(\frac{\pi}{6}\right)+1\times\sqrt{2}\cos\left(\frac{3\pi}{6}\right)\right.$$
$$\left.+1\times\sqrt{2}\cos\left(\frac{5\pi}{6}\right)\right\}$$
$$=\frac{\sqrt{2}}{3}\left\{1\times\frac{\sqrt{3}}{2}+1\times0+1\times\left(-\frac{\sqrt{3}}{2}\right)\right\}=0$$

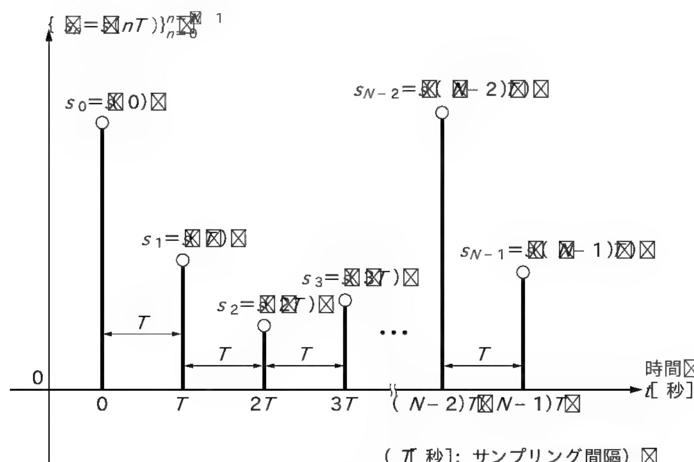


図24.5　サンプル数N個のディジタル信号波形



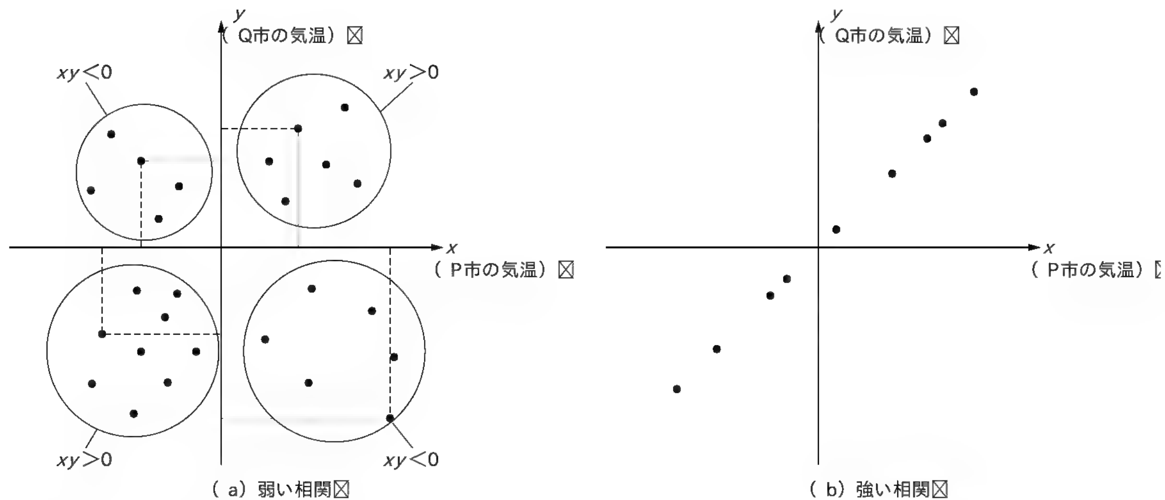


図24.6　P市とQ市の気温の相互相関

$$\left\langle \boldsymbol{\phi}^{(1)}, \boldsymbol{\phi}^{(2)} \right\rangle = \frac{1}{3}\left\{ \sqrt{2}\cos\left(\frac{\pi}{6}\right) \times \sqrt{2}\cos\left(\frac{2\pi}{6}\right) \right.$$

$$\left. \sqrt{2}\cos\left(\frac{3\pi}{6}\right) \times \sqrt{2}\cos\left(\frac{6\pi}{6}\right) + \sqrt{2}\cos\left(\frac{5\pi}{6}\right) \times \sqrt{2}\cos\left(\frac{10\pi}{6}\right) \right\}$$

$$= \frac{2}{3}\left\{ \frac{\sqrt{3}}{2} \times \frac{1}{2} + 0 \times (-1) + \left(-\frac{\sqrt{3}}{2}\right) \times \frac{1}{2} \right\} = 0$$

● **信号値$\{s_n\}_{n=0}^{n=2}$から展開係数$\{G_\ell\}_{\ell=0}^{\ell=2}$を算出**

式(7)に基づき，N=3として式(8)を代入することにより，

$$G_0 = \left\langle \boldsymbol{s}, \boldsymbol{\phi}^{(0)} \right\rangle = \frac{1}{3}(s_0 + s_1 + s_2) \quad \cdots\cdots (9)$$

$$G_1 = \left\langle \boldsymbol{s}, \boldsymbol{\phi}^{(1)} \right\rangle$$

$$= \frac{1}{3}\left[s_0 \times \left\{ \sqrt{2}\cos\left(\frac{\pi}{6}\right) \right\} + s_1 \times \left\{ \sqrt{2}\cos\left(\frac{3\pi}{6}\right) \right\} + s_2 \times \left\{ \sqrt{2}\cos\left(\frac{5\pi}{6}\right) \right\} \right]$$

$$= \frac{1}{3}\left[\frac{\sqrt{6}}{2}s_0 - \frac{\sqrt{6}}{2}s_2 \right] \quad \cdots\cdots (10)$$

$$G_2 = \left\langle \boldsymbol{s}, \boldsymbol{\phi}^{(2)} \right\rangle$$

$$= \frac{1}{3}\left[s_0 \times \left\{ \sqrt{2}\cos\left(\frac{2\pi}{6}\right) \right\} + s_1 \times \left\{ \sqrt{2}\cos\left(\frac{6\pi}{6}\right) \right\} + s_2 \times \left\{ \sqrt{2}\cos\left(\frac{10\pi}{6}\right) \right\} \right]$$

$$= \frac{1}{3}\left[\frac{\sqrt{2}}{2}s_0 - \sqrt{2}s_1 + \frac{\sqrt{2}}{2}s_2 \right] \quad \cdots\cdots (11)$$

● **展開係数$\{G_\ell\}_{\ell=0}^{\ell=2}$から信号値$\{s_n\}_{n=0}^{n=2}$を算出**

式(6)に基づき，N=3として式(8)を代入することにより，信号値$\{s_n\}_{n=0}^{n=2}$は次のように計算される．

$$s_0 = G_0 + \sqrt{2}\cos\left(\frac{\pi}{6}\right)G_1 + \sqrt{2}\cos\left(\frac{2\pi}{6}\right)G_2$$

$$= G_0 + \frac{\sqrt{6}}{2}G_1 + \frac{\sqrt{2}}{2}G_2 \quad \cdots\cdots (12)$$

$$s_1 = G_0 + \sqrt{2}\cos\left(\frac{3\pi}{6}\right)G_1 + \sqrt{2}\cos\left(\frac{6\pi}{6}\right)G_2$$

$$= G_0 - \sqrt{2}G_2 \quad \cdots\cdots (13)$$

$$s_2 = G_0 + \sqrt{2}\cos\left(\frac{5\pi}{6}\right)G_1 + \sqrt{2}\cos\left(\frac{10\pi}{6}\right)G_2$$

$$= G_0 - \frac{\sqrt{6}}{2}G_1 + \frac{\sqrt{2}}{2}G_2 \quad \cdots\cdots (14)$$

信号相関と正規直交表示

まず，統計的なデータ処理技術に用いられる信号相関という概念の説明から始めよう(実は，フーリエ変換は信号波と正弦波との相関値なのだが)．

たとえば，P市とQ市の気温をそれぞれx[℃]，y[℃]とし，図24.6のようにxとyの測定値がxy平面上の点として与えられるものとしよう．図24.6(a)のほうはxとyの間にある程度の弱い相関があることがわかり，図24.6(b)からはxとyとの間には強い相関(傾きが+1で45度の直線上に点が存在)があることがわかる．

これを定量的に表現するとなると，よく知られているようにN個の各点に対するxとyの積の総和として，

$$R = \sum_{n=0}^{N-1} x_n y_n \quad \cdots\cdots (15)$$

に注目する．たとえば点が第1，第3象限にあると積は正であり，第2，第4象限にあると積は負である．もし，点がxy平面上に一様に散らばっていて，xとyの積が正になったり負にな

ると，式(15)の値は小さくなって相関が少ないことを意味する．また，積 $x_n y_n$ がつねに正であれば正の相関があり，負のときは負の相関があると考えられる．したがって，各点に関する積 $x_n y_n$ の総和の違いで相関の大小を推定することができよう．ただし，このままでは点の数が増加すると大きな値になるので，通常は正規化する必要がある．ふつうは，1サンプルあたりに換算した値として，

$$R=\frac{1}{N}\sum_{n=0}^{N-1}x_n y_n \quad\cdots\cdots(16)$$

で相関値を定義する．とくに，われわれが興味深いものは，x と y がディジタル信号波形で，たとえば x が正弦波信号 $\boldsymbol{f}=\{f_n\}_{n=0}^{n=N-1}$，$y$ があるシステムの出力信号波形 $\boldsymbol{g}=\{g_n\}_{n=0}^{n=N-1}$ であろう．このような場合，信号波形が正弦波とどの程度の相関があるのかを知りたいことも多い(実は，信号波形と正弦波の相関がフーリエ変換そのものであるのだが…)．

ここでは，信号は通常実数であるが，より一般的に複素数とし，式(16)の相関値の定義を適用するとすれば，

$$R=\frac{1}{N}\sum_{n=0}^{N-1}f_n\overline{g_n} \quad\cdots\cdots(17)$$

と表される．ただし，変数の上部に付けた"　　"は複素共役を示す．

また，式(2)や式(7)の内積で表示すると，

$$R=\frac{1}{N}\sum_{n=0}^{N-1}f_n\overline{g_n}=\langle \boldsymbol{f},\boldsymbol{g}\rangle \quad\cdots\cdots(18)$$

とあることが導かれる．式(18)において，

$$\begin{cases}\boldsymbol{f}=\boldsymbol{s} & ;\ \text{ディジタル信号}\\ \boldsymbol{g}=\boldsymbol{\phi}^{(\ell)} & ;\ \text{正規直交基底ベクトル}\end{cases} \quad\cdots\cdots(19)$$

と見なせば，式(7)より，

$$R=\langle \boldsymbol{s},\boldsymbol{\phi}^{(\ell)}\rangle=G_\ell \quad\cdots\cdots(20)$$

となり，相関値と正規直交基底ベクトルによる展開係数とが見事に一致するのである．

正規直交基底とDFT

いま，正規直交基底ベクトル $\{\boldsymbol{\phi}^{(\ell)}\}_{\ell=0}^{\ell=N-1}$ を，

$$\begin{aligned}\boldsymbol{\phi}^{(\ell)}&=\{1\ \ V_N^{\ \ell}\ \ V_N^{\ 2\ell}\ \cdots\ V_N^{\ \ell(N-1)}\}\\ &=\{V_N^{\ \ell n}\}_{n=0}^{n=N-1} \quad\cdots\cdots(21)\end{aligned}$$

ただし，$V_N=\mathrm{e}^{j\frac{2\pi}{N}}$ ……(22)

とするとき，式(21)と式(22)を式(7)に代入することにより，N サンプルに対する展開係数は，

$$G_\ell=\langle \boldsymbol{s},\boldsymbol{\phi}^{(\ell)}\rangle=\frac{1}{N}\sum_{n=0}^{N-1}s_n\overline{V_N^{\ \ell n}} \quad\cdots\cdots(23)$$

と表される．また，

$$\overline{V_N^{\ \ell n}}=\overline{\left(\mathrm{e}^{j\frac{2\pi}{N}}\right)^{\ell n}}=\left(\mathrm{e}^{-j\frac{2\pi}{N}}\right)^{\ell n} \quad\cdots\cdots(24)$$

であるので，

$$W_N=\mathrm{e}^{-j\frac{2\pi}{N}} \quad\cdots\cdots(25)$$

と置けば，

$$\overline{V_N^{\ \ell n}}=W_N^{\ \ell n} \quad\cdots\cdots(26)$$

となる関係があることから，式(23)は，

$$G_\ell=\langle \boldsymbol{s},\boldsymbol{\phi}^{(\ell)}\rangle=\frac{1}{N}\sum_{n=0}^{N-1}s_nW_N^{\ \ell n} \quad\cdots\cdots(27)$$

と表され，DFT計算式に相当することがわかる．

次に，式(6)に式(21)を代入すれば，

$$s_n=\sum_{\ell=0}^{N-1}G_\ell V_N^{\ \ell n} \quad\cdots\cdots(28)$$

となり，さらに式(22)と式(25)より，

$$V_N^{\ \ell n}=W_N^{\ -\ell n} \quad\cdots\cdots(29)$$

であるので，

$$s_n=\sum_{\ell=0}^{N-1}G_\ell W_N^{\ -\ell n} \quad\cdots\cdots(30)$$

と表され，IDFT(ディジタル逆フーリエ変換)計算式が導かれる．

例題2

サンプル数 N=3に対するDFTとIDFTの計算式を示せ．

解答2

まず，DFTの正規直交基底ベクトル $\{\boldsymbol{\phi}^{(\ell)}\}_{\ell=0}^{\ell=2}$ は N=3として $V_3=\mathrm{e}^{j\frac{2\pi}{3}}$，なので，

$$\begin{cases}\boldsymbol{\phi}^{(0)}=[1\quad 1\quad 1]\\ \boldsymbol{\phi}^{(1)}=[1\quad V_3^{\ 1}\quad V_3^{\ 2}]\\ \boldsymbol{\phi}^{(2)}=[1\quad V_3^{\ 2}\quad V_3^{\ 4}]\end{cases} \quad\cdots\cdots(31)$$

と表される．なお，$\{\boldsymbol{\phi}^{(\ell)}\}_{\ell=0}^{\ell=2}$ が正規直交性(式(3))を有することは各自で検証しておいてほしい．

次に，DFT計算式は，式(23)に式(31)を代入し，式(26)を適用することにより，

$$G_0=\langle \boldsymbol{s},\boldsymbol{\phi}^{(0)}\rangle=\frac{1}{3}(s_0+s_1+s_2) \quad\cdots\cdots(32)$$

$$\begin{aligned}G_1=\langle \boldsymbol{s},\boldsymbol{\phi}^{(1)}\rangle&=\frac{1}{3}\left(s_0+s_1\overline{V_3^{\ 1}}+s_2\overline{V_3^{\ 2}}\right)\\ &=\frac{1}{3}\left(s_0+s_1W_3^{\ 1}+s_2W_3^{\ 2}\right) \quad\cdots\cdots(33)\end{aligned}$$

$$\begin{aligned}G_2=\langle \boldsymbol{s},\boldsymbol{\phi}^{(2)}\rangle&=\frac{1}{3}\left(s_0+s_1\overline{V_3^{\ 2}}+s_2\overline{V_3^{\ 4}}\right)\\ &=\frac{1}{3}\left(s_0+s_1W_3^{\ 2}+s_2W_3^{\ 4}\right) \quad\cdots\cdots(34)\end{aligned}$$

となり，よく見慣れたDFT計算式が得られるのである．さらに，式(33)と式(34)の計算を進めると，

$$\begin{aligned}G_1&=\frac{1}{3}\left[s_0+s_1\mathrm{e}^{-j\frac{2\pi}{3}}+s_2\mathrm{e}^{-j\frac{4\pi}{3}}\right]\\ &=\frac{1}{3}\left[s_0+s_1\times\left\{\cos\left(\frac{2\pi}{3}\right)-j\sin\left(\frac{2\pi}{3}\right)\right\}\right.\\ &\qquad\left.+s_2\times\left\{\cos\left(\frac{4\pi}{3}\right)-j\sin\left(\frac{4\pi}{3}\right)\right\}\right]\end{aligned}$$



$$=\frac{1}{3}\left[s_0+\left(-\frac{1}{2}-j\frac{\sqrt{3}}{2}\right)s_1+\left(-\frac{1}{2}+j\frac{\sqrt{3}}{2}\right)s_2\right] \cdots\cdots (35)$$

$$G_2=\frac{1}{3}\left[s_0+s_1 e^{-j\frac{4\pi}{3}}+s_2 e^{-j\frac{8\pi}{3}}\right]$$

$$=\frac{1}{3}\left[s_0+s_1\times\left\{\cos\left(\frac{4\pi}{3}\right)-j\sin\left(\frac{4\pi}{3}\right)\right\}\right.$$

$$\left.+s_2\times\left\{\cos\left(\frac{8\pi}{3}\right)-j\sin\left(\frac{8\pi}{3}\right)\right\}\right]$$

$$=\frac{1}{3}\left[s_0+\left(-\frac{1}{2}+j\frac{\sqrt{3}}{2}\right)s_1+\left(-\frac{1}{2}-j\frac{\sqrt{3}}{2}\right)s_2\right]=\overline{G_1}$$

$$\cdots\cdots (36)$$

一方，式(28)あるいは式(30)において，サンプル数N=3の例では，式(31)の各要素を代入すればよいので，

$$\begin{cases} s_0=G_0+G_1+G_2 \\ s_1=G_0+G_1V_3^{1}+G_2V_3^{2} \\ s_2=G_0+G_1V_3^{2}+G_2V_3^{4} \end{cases} \cdots\cdots (37)$$

となり，さらに式(29)の関係を考慮して書き換えると，

$$\begin{cases} s_0=G_0+G_1+G_2 \\ s_1=G_0+G_1W_3^{-1}+G_2W_3^{-2} \\ s_2=G_0+G_1W_3^{-2}+G_2W_3^{-4} \end{cases} \cdots\cdots (38)$$

と表される．導き出した式をよく見てもらえれば，なんとIDFT(ディジタル逆フーリエ変換)が得られている(お見事でしょ)．

つまり，式(5)の正規直交基底ベクトルによるディジタル信号の展開そのものがIDFTだというわけである．別な見かたをすれば，式(32)～式(34)を3個の未知変数s_0，s_1，s_2に関する連立方程式に見立てて解いた結果が式(38)に一致するのである．読者のみなさんにはぜひ一度確かめてみてもらいたい．

以上のように，正規直交基底ベクトルによるディジタル信号の表現とDFTがまったく同じことなんだという点をしっかりと理解していただきたい(もちろん，DCTもしかりである)．

DFTとDCTの関係

いま，一般性を損なうことなくサンプル数Nを奇数とし，DFT計算[式(23)あるいは式(27)]に基づき，整数ℓを$\ell/2$に置き換えて，

$$X_\ell=\left\langle s, \phi^{(\ell)}\right\rangle=G_{\frac{\ell}{2}} \cdots\cdots (39)$$

$$\phi^{(\ell)}=\left\{1\quad V_N^{\frac{\ell}{2}}\quad V_N^{\ell}\quad\cdots\quad V_N^{\frac{\ell(N-1)}{2}}\right\}$$

$$=\left\{V_N^{\frac{\ell n}{2}}\right\}_{n=0}^{n=N-1};\ \ell=0,1,2,\cdots,(N-1) \cdots\cdots (40)$$

と表すことにする．そこで，直流成分以外($\ell\neq 0$)では式(39)にちょいと「おまじない」を掛けて，

$$X_\ell=\frac{1}{2}\times\left\langle 2s, \phi^{(\ell)}\right\rangle$$

$$=\frac{1}{2}\times\left\langle (s+s), \phi^{(\ell)}\right\rangle \cdots\cdots (41)$$

と変形する．さらに，恒等的にゼロ(0)になる内積，すなわち，

$$\frac{1}{2}\times\left\langle (s-s), {}^{*}\phi^{(\ell)}\right\rangle \cdots\cdots (42)$$

ただし，

$${}^{*}\phi^{(\ell)}=\left\{V_N^{\frac{(2N-1)\ell}{2}}\cdots V_N^{\frac{(N+2)\ell}{2}}\quad V_N^{\frac{(N+1)\ell}{2}}\quad V_N^{\frac{N\ell}{2}}\right\}$$

$$=\left\{V_N^{\frac{(2N-n-1)\ell}{2}}\right\}_{n=0}^{n=N-1} \cdots\cdots (43)$$

を式(41)の右辺に加えても不変なので，

$$X_\ell=\frac{1}{2}\times\left\{\left\langle (s+s), \phi^{(\ell)}\right\rangle+\underbrace{\left\langle (s-s), {}^{*}\phi^{(\ell)}\right\rangle}_{0}\right\} \cdots\cdots (44)$$

となる関係が得られる．

ところで，二つの正規直交基底ベクトル$\left\{\phi^{(\ell)}\right\}_{\ell=0}^{\ell=N-1}$と$\left\{{}^{*}\phi^{(\ell)}\right\}_{\ell=0}^{\ell=N-1}$はそれぞれ，次のように変形して別表現することができる．つまり，

$$V_{2N}=e^{j\frac{2\pi}{2N}} \cdots\cdots (45)$$

であるので，式(40)より，

$$V_N=V_{2N}^{\,2} \cdots\cdots (46)$$

となる関係が成立して，式(40)は，

$$\phi^{(\ell)}=\left\{V_{2N}^{\,\ell n}\right\}_{n=0}^{n=N-1}$$

$$=V_{2N}^{\,-\frac{\ell}{2}}\times\left\{V_{2N}^{\,\frac{(2n+1)\ell}{2}}\right\}_{n=0}^{n=N-1} \cdots\cdots (47)$$

と表される．

同様に，式(46)の関係を式(43)に適用すれば，

$${}^{*}\phi^{(\ell)}=\left\{V_{2N}^{\,(2N-n-1)\ell}\right\}_{n=0}^{n=N-1} \cdots\cdots (48)$$

となり，

$$V_{2N}^{\,2N}=\left(e^{j\frac{2\pi}{2N}}\right)^{2N}=e^{j2\pi}$$

$$=\underbrace{\cos(2\pi)}_{1}+j\underbrace{\sin(2\pi)}_{0}=1 \cdots\cdots (49)$$

であり，

$$V_{2N}^{\,2N\ell}=1\ ;\ \ell\text{ は整数} \cdots\cdots (50)$$

となる関係が成り立つので，式(48)は，

$${}^{*}\phi^{(\ell)}=\left\{V_{2N}^{\,-(n+1)\ell}\right\}_{n=0}^{n=N-1}$$

$$=V_{2N}^{\,-\frac{\ell}{2}}\times\left\{V_{2N}^{\,-\frac{(2n+1)\ell}{2}}\right\}_{n=0}^{n=N-1} \cdots\cdots (51)$$

と表される．ここで，式(47)と式(51)において，

$$\psi^{(\ell)}=\left\{\phi_n^{(\ell)}\right\}_{n=0}^{n=N-1}=\left\{V_{2N}^{\,\frac{(2n+1)\ell}{2}}\right\}_{n=0}^{n=N-1} \cdots\cdots (52)$$

と置けば，式(52)の複素共役をとることにより，

$$\overline{\psi^{(\ell)}}=\left\{\overline{V_{2N}^{\,\frac{(2n+1)\ell}{2}}}\right\}_{n=0}^{n=N-1}=\left\{V_{2N}^{\,-\frac{(2n+1)\ell}{2}}\right\}_{n=0}^{n=N-1} \cdots\cdots (53)$$

と表されることになる．式(52)と式(53)を用いて，正規直交

基底ベクトルの要素 $\phi^{(\ell)}$，${}^*\phi^{(\ell)}$はそれぞれ，

$$\phi^{(\ell)} = V_{2N}^{-\frac{\ell}{2}} \times \Psi^{(\ell)} \quad \cdots\cdots (54)$$

$${}^*\phi^{(\ell)} = V_{2N}^{-\frac{\ell}{2}} \times \overline{\Psi^{(\ell)}} \quad \cdots\cdots (55)$$

となる．

以上の計算結果に基づき，式(54)と式(55)を式(44)に代入することにより，

$$\begin{aligned} X_\ell &= \frac{1}{2}\times\left\{\left\langle s, V_{2N}^{-\frac{\ell}{2}}\Psi^{(\ell)}\right\rangle + \left\langle s, V_{2N}^{-\frac{\ell}{2}}\Psi^{(\ell)}\right\rangle \right. \\ &\qquad \left. + \left\langle s, V_{2N}^{-\frac{\ell}{2}}\overline{\Psi^{(\ell)}}\right\rangle - \left\langle s, V_{2N}^{-\frac{\ell}{2}}\overline{\Psi^{(\ell)}}\right\rangle\right\} \\ &= V_{2N}^{\frac{\ell}{2}} \times \frac{1}{2} \times \left\{\left\langle s, \Psi^{(\ell)}\right\rangle + \left\langle s, \Psi^{(\ell)}\right\rangle \right. \\ &\qquad \left. + \left\langle s, \overline{\Psi^{(\ell)}}\right\rangle - \left\langle s, \overline{\Psi^{(\ell)}}\right\rangle\right\} \quad \cdots\cdots (56) \end{aligned}$$

となる．さらに，

$$X_\ell = V_{2N}^{\frac{\ell}{2}} \times \tilde{X}_\ell \quad \cdots\cdots (57)$$

$$\begin{aligned} \tilde{X}_\ell &= \frac{1}{2}\times\left\{\left\langle s, \Psi^{(\ell)}\right\rangle + \left\langle s, \overline{\Psi^{(\ell)}}\right\rangle\right\} \\ &\quad + \frac{1}{2}\times\left\{\left\langle s, \Psi^{(\ell)}\right\rangle - \left\langle s, \overline{\Psi^{(\ell)}}\right\rangle\right\} \\ &= \left\langle s, \frac{\Psi^{(\ell)}}{2}\right\rangle + \left\langle s, \frac{\overline{\Psi^{(\ell)}}}{2}\right\rangle + \left\langle s, \frac{\Psi^{(\ell)}}{2}\right\rangle + \left\langle s, -\frac{\overline{\Psi^{(\ell)}}}{2}\right\rangle \\ &= \left\langle s, \frac{\Psi^{(\ell)} + \overline{\Psi^{(\ell)}}}{2}\right\rangle + \left\langle s, \frac{\Psi^{(\ell)} - \overline{\Psi^{(\ell)}}}{2}\right\rangle \end{aligned}$$

$$\cdots\cdots (58)$$

と変形できる．ここで，$\left(\Psi^{(\ell)} + \overline{\Psi^{(\ell)}}\right)/2$の各要素($n = 0, 1, 2, \cdots, (N-1)$)は，

$$\begin{aligned} \frac{\Psi_n^{(\ell)} + \overline{\Psi_n^{(\ell)}}}{2} &= \frac{V_{2N}^{\frac{(2n+1)\ell}{2}} + V_{2N}^{-\frac{(2n+1)\ell}{2}}}{2} \\ &= \frac{e^{j\frac{(2n+1)\ell}{2N}\pi} + e^{-j\frac{(2n+1)\ell}{2N}\pi}}{2} \\ &= \cos\left[\frac{(2n+1)\ell}{2N}\pi\right] \quad \cdots\cdots (59) \end{aligned}$$

であるので，内積の定義を適用することにより，

$$\left\langle s, \frac{\Psi^{(\ell)} + \overline{\Psi^{(\ell)}}}{2}\right\rangle = \frac{1}{N}\sum_{n=0}^{N-1} s_n \cos\left[\frac{(2n+1)\ell}{2N}\pi\right] \quad \cdots\cdots (60)$$

と表される．また，$\left(\Psi^{(\ell)} + \overline{\Psi^{(\ell)}}\right)/2$の各要素($n = 0, 1, 2, \cdots, (N-1)$)は，

$$\begin{aligned} \frac{\Psi_n^{(\ell)} - \overline{\Psi_n^{(\ell)}}}{2} &= \frac{V_{2N}^{\frac{(2n+1)\ell}{2}} - V_{2N}^{-\frac{(2n+1)\ell}{2}}}{2} \\ &= \frac{e^{j\frac{(2n+1)\ell}{2N}\pi} - e^{-j\frac{(2n+1)\ell}{2N}\pi}}{2} \\ &= j\sin\left[\frac{(2n+1)\ell}{2N}\pi\right] \quad \cdots\cdots (61) \end{aligned}$$

であるので，内積の定義を適用することにより，

$$\left\langle s, \frac{\Psi^{(\ell)} - \overline{\Psi^{(\ell)}}}{2}\right\rangle = -j\frac{1}{N}\sum_{n=0}^{N-1} s_n \sin\left[\frac{(2n+1)\ell}{2N}\pi\right] \quad \cdots\cdots (62)$$

と導かれる．なお，式(59)と式(61)の導出に際しては，オイラーの公式，すなわち，

$$\frac{e^{j\theta} + e^{-j\theta}}{2} = \cos(\theta) \quad \cdots\cdots (63)$$

$$\frac{e^{j\theta} - e^{-j\theta}}{2} = j\sin(\theta) \quad \cdots\cdots (64)$$

を用いている．

また，くどいようだが，式(56)，式(62)の計算においては，内積の定義における“複素共役をとって計算する”点に十分な注意が必要である．

以上より，式(58)は，

$$\begin{aligned} \tilde{X}_\ell &= \frac{1}{N}\sum_{n=0}^{N-1} s_n \cos\left[\frac{(2n+1)\ell}{2N}\pi\right] \\ &\qquad - j\frac{1}{N}\sum_{n=0}^{N-1} s_n \sin\left[\frac{(2n+1)\ell}{2N}\pi\right] \\ &= \frac{1}{\sqrt{2}} \times \frac{1}{N}\sum_{n=0}^{N-1} s_n \left\{\sqrt{2}\cos\left[\frac{(2n+1)\ell}{2N}\pi\right]\right\} \\ &\qquad - j\frac{1}{\sqrt{2}} \times \frac{1}{N}\sum_{n=0}^{N-1} s_n \left\{\sqrt{2}\sin\left[\frac{(2n+1)\ell}{2N}\pi\right]\right\} \end{aligned}$$

$$\cdots\cdots (65)$$

と整理される．ここで，

$$\begin{cases} C_\ell = \dfrac{1}{N}\displaystyle\sum_{n=0}^{N-1} s_n \left\{\sqrt{2}\cos\left[\dfrac{(2n+1)\ell}{2N}\pi\right]\right\} & \cdots\cdots (66) \\ S_\ell = \dfrac{1}{N}\displaystyle\sum_{n=0}^{N-1} s_n \left\{\sqrt{2}\sin\left[\dfrac{(2n+1)\ell}{2N}\pi\right]\right\} & \cdots\cdots (67) \end{cases}$$

と置けば，$\{C_\ell\}_{\ell=0}^{\ell=N-1}$ は DCT 値，$\{S_\ell\}_{\ell=0}^{\ell=N-1}$ は DST 値に相当する．以上の結果に基づき，式(39)，式(57)，式(65)～式(67)と見比べることにより，

$$G_{\frac{\ell}{2}} = \frac{1}{\sqrt{2}} V_{2N}^{\frac{\ell}{2}} \left[C_\ell - jS_\ell\right] \;\; ; \ell = 2, 4, 6, \cdots, 2(N-1) \quad \cdots (68)$$

で表される関係が導かれる．見事に，DFT と DCT および DST［ディジタル・サイン変換といい，式(67)に相当］との関係が得られたわけである．

なお，直流成分($\ell=0$)では，式(67)より $S_0=0$であり，また式(21)と式(40)はまったく同じ値になるので，式(23)あるいは式(27)に $\ell=0$を代入したものに一致することは明白である．すなわち，

$$G_0 = X_0 = C_0 = \frac{1}{N}\sum_{n=0}^{N-1} s_n \quad \cdots\cdots (69)$$

であり，式(66)と式(69)を一つにまとめると，

$$C_\ell = \frac{1}{N}\sum_{n=0}^{N-1} \gamma_\ell s_n \cos\left[\frac{(2n+1)\ell}{2N}\pi\right] \quad \cdots\cdots (70)$$



直流（$\ell=0$）	DFT	DCT	DST
	$G_0=C_0$		$S_0=0$
交流（$\ell=2, 4, \cdots, (N-1)$）	DFT	DCT	DST
	$G_{\frac{\ell}{2}}=\frac{1}{\sqrt{2}} V_{2N}^{\frac{\ell}{4}}\{C_\ell - jS_\ell\}$ $C_\ell=\frac{1}{N}\sum_{n=0}^{N-1} s_n \times \left(\sqrt{2}\cos\left\{\frac{(2n+1)\ell}{2N}\pi\right\}\right)$ $S_\ell=\frac{1}{N}\sum_{n=0}^{N-1} s_n \times \left(\sqrt{2}\sin\left\{\frac{(2n+1)\ell}{2N}\pi\right\}\right)$ 正規直交基底ベクトルの要素		

図 24.7　DFT，DCT，DSTの相互関係

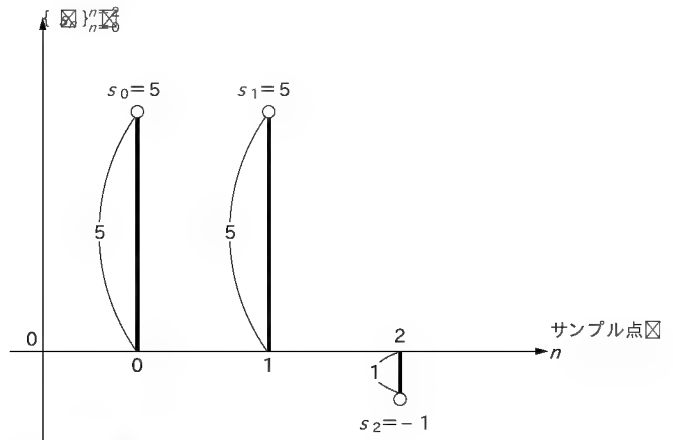


図 24.8　例題3のディジタル信号波形

ただし，$\gamma_\ell=\begin{cases}1 ; \ell=0 \\ \sqrt{2} ; \ell\neq 0\end{cases}$ ……………………（71）

で表される DCT の定義式が得られる．式（71）より，サンプル数 N における DCT の正規直交基底ベクトル $\{\boldsymbol{\phi}^{(\ell)}\}_{\ell=0}^{\ell=N-1}$ は，

$$\boldsymbol{\phi}^{(\ell)}=\left\{1 \quad \sqrt{2}\cos\left[\frac{\pi}{2N}\right] \quad \sqrt{2}\cos\left[\frac{3\pi}{2N}\right] \quad \cdots, \cdots, \sqrt{2}\cos\left[\frac{(2N-1)\pi}{2N}\right]\right\}$$

$$=\left\{\gamma_\ell \cos\left[\frac{(2n+1)\ell}{2N}\pi\right]\right\}_{n=0}^{n=N-1} \quad \cdots\cdots\cdots(72)$$

であることがわかる．

逆に，式（6）において，G_ℓ を C_ℓ と読み替えれば，

$$s_n=\sum_{\ell=0}^{N-1}\gamma_\ell C_\ell \cos\left[\frac{(2n+1)\ell}{2N}\pi\right] \quad \cdots\cdots\cdots(73)$$

と表され，IDCT（ディジタル逆コサイン変換）の計算式が得られる．

また，式（68）と式（69）より，DFT 値，DCT 値，DST 値の相互関係は，

$$G_{\frac{\ell}{2}}=\frac{1}{\gamma_\ell}V_{2N}^{\frac{\ell}{2}}\left[C_\ell - jS_\ell\right]\ ;\ \ell=0, 2, 4, \cdots, (N-1) \quad \cdots\cdots(74)$$

と統一した形式で表される（図 24.7）．

例題3

図 24.8 に示すディジタル信号に対する DFT 値，DCT 値，DST 値を求め，式（74）の関係が成立することを示せ．

解答3

まず，それぞれの変換値を求める．

● DFT 値〔式（32），式（33）による〕

$$G_0=\frac{1}{3}[5+5+(-1)]=3 \quad \cdots\cdots\cdots(75)$$

$$G_1=\frac{1}{3}\left[5+5\times\left(-\frac{1}{2}-j\frac{\sqrt{3}}{2}\right)+(-1)\times\left(-\frac{1}{2}+j\frac{\sqrt{3}}{2}\right)\right]$$

$$=1-j\sqrt{3}=2e^{-j\frac{\pi}{3}} \quad \cdots\cdots\cdots(76)$$

$$G_2=\frac{1}{3}\left[5+5\times\left(-\frac{1}{2}+j\frac{\sqrt{3}}{2}\right)+(-1)\times\left(-\frac{1}{2}-j\frac{\sqrt{3}}{2}\right)\right]$$

$$=1+j\sqrt{3}=2e^{j\frac{\pi}{3}}=\overline{G_1} \quad \cdots\cdots\cdots(77)$$

● DCT 値〔式（70）による〕

式（9）～式（11）の G_ℓ を C_ℓ に読み替えた値に一致する．

$$C_0=\frac{1}{3}[5+5+(-1)]=3 \quad \cdots\cdots\cdots(78)$$

$$C_1=\frac{1}{3}\left[\frac{\sqrt{6}}{2}\times 5-\frac{\sqrt{6}}{2}\times(-1)\right]=\sqrt{6} \quad \cdots\cdots\cdots(79)$$

$$C_2=\frac{1}{3}\left[\frac{\sqrt{2}}{2}\times 5-\sqrt{2}\times 5+\frac{\sqrt{2}}{2}\times(-1)\right]=-\sqrt{2} \quad \cdots\cdots(80)$$

● DST 値〔式（67）による〕

$$S_0=\frac{1}{3}[0+0+0]=0 \quad \cdots\cdots\cdots(81)$$

$$S_1=\frac{1}{3}\left[\frac{\sqrt{2}}{2}\times 5+\sqrt{2}\times 5+\frac{\sqrt{2}}{2}\times(-1)\right]=\frac{7\sqrt{2}}{3} \quad \cdots\cdots(82)$$

$$S_2=\frac{1}{3}\left[\frac{\sqrt{6}}{2}\times 5-\frac{\sqrt{6}}{2}\times(-1)\right]=\sqrt{6} \quad \cdots\cdots\cdots(83)$$

以上の結果より，$\ell=0$，2 に対して，式（74）の関係が成立することを調べる．$\ell=0$ の場合は明白で，式（75），式（78），式（81）より，

$$G_0=C_0 \quad \cdots\cdots\cdots(84)$$

となる．また，$\ell=2$ の場合は，式（74），式（76），式（80），式（83）より，

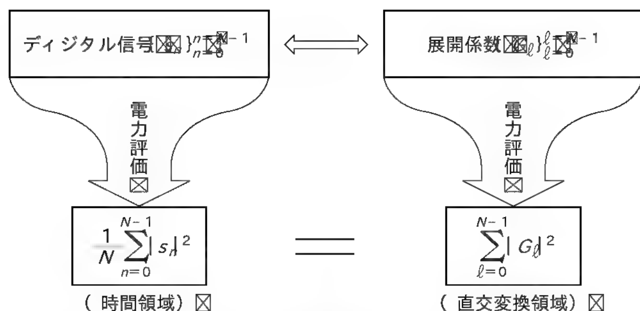


図 24.9　パーシバルの定理による電力評価

$$\frac{1}{\sqrt{2}}V_6^1\left[C_2 - jS_2\right] = \frac{1}{\sqrt{2}}e^{j\frac{2\pi}{6}} \times \left[-\sqrt{2} - j\sqrt{6}\right]$$

↓（横座標で表す）

$$= \frac{1}{\sqrt{2}}e^{j\frac{2\pi}{6}} \times 2\sqrt{2}e^{-j\frac{2\pi}{3}}$$

$$= 2e^{-j\frac{\pi}{3}} = G_1 \quad \cdots\cdots\cdots\cdots (85)$$

となる．

パーシバルの定理

ところで，N サンプルのディジタル信号 $\{s_n\}_{n=0}^{n=N-1}$ の平均電力は，

$$P = \frac{1}{N}\sum_{n=0}^{N-1} s_n \overline{s_n} \quad \cdots\cdots\cdots\cdots (86)$$

であり，内積で表せば，

$$P = \langle \boldsymbol{s}, \boldsymbol{s} \rangle \quad \cdots\cdots\cdots\cdots (87)$$

となり，1 サンプルあたりの平均エネルギに相当する．さらに，

$$s_n \overline{s_n} = |s_n|^2 \quad \cdots\cdots\cdots\cdots (88)$$

となる関係を式(86)に適用すると，

$$P = \frac{1}{N}\sum_{n=0}^{N-1} |s_n|^2 \quad \cdots\cdots\cdots\cdots (89)$$

となる別表現も可能であり，ディジタル信号値の絶対値の 2 乗和に等しい．

そこで，ディジタル信号 s を正規直交基底ベクトル $\{\boldsymbol{\phi}^{(\ell)}\}_{\ell=0}^{\ell=N-1}$ で展開した表現式，すなわち，

$$\boldsymbol{s} = \sum_{\ell=0}^{N-1} G_\ell \boldsymbol{\phi}^{(\ell)} \quad \cdots\cdots\cdots\cdots [\text{式(5)の再掲}]$$

を式(87)に代入してみよう．すると，

$$P = \left\langle \sum_{k=0}^{N-1} G_k \boldsymbol{\phi}^{(k)}, \sum_{\ell=0}^{N-1} G_\ell \boldsymbol{\phi}^{(\ell)} \right\rangle$$

$$= \sum_{k=0}^{N-1}\sum_{\ell=0}^{N-1} G_k \overline{G_\ell} \left\langle \boldsymbol{\phi}^{(k)}, \boldsymbol{\phi}^{(\ell)} \right\rangle \quad \cdots\cdots\cdots\cdots (90)$$

となり，式(3)の直交性の関係を適用することにより，$k \neq \ell$ のときは内積がゼロ(0)になるので，

$$P = \sum_{\ell=0}^{N-1} G_\ell \overline{G_\ell} \left\langle \boldsymbol{\phi}^{(\ell)}, \boldsymbol{\phi}^{(\ell)} \right\rangle \quad \cdots\cdots\cdots\cdots (91)$$

と簡略化される．続いて，式(3)の正規性を適用することにより，

$$P = \sum_{\ell=0}^{N-1} |G_\ell|^2 \underbrace{\left\langle \boldsymbol{\phi}^{(\ell)}, \boldsymbol{\phi}^{(\ell)} \right\rangle}_{1}$$

$$= \sum_{\ell=0}^{N-1} |G_\ell|^2 \quad \cdots\cdots\cdots\cdots (92)$$

のように，展開係数(たとえば DFT 値，あるいは DCT 値に相当)の絶対値の 2 乗和として表される．

よって，式(89)と式(92)より，

$$\frac{1}{N}\sum_{n=0}^{N-1} |s_n|^2 = \sum_{\ell=0}^{N-1} |G_\ell|^2 \quad \cdots\cdots\cdots\cdots (93)$$

となるので，時間領域の信号値 $\{s_n\}_{n=0}^{n=N-1}$ でも直交変換領域の展開係数 $\{G_\ell\}_{\ell=0}^{\ell=N-1}$ でも区別することなく，電力を評価することができることがわかる(図 24.9)．この性質は，たとえば雑音を取り扱うときには非常に便利である．

例題4

例題3 のディジタル信号(図 24.8)に対する直交変換領域の展開係数が DFT 値，DCT 値とするとき，平均電力について式(93)の関係が成立することを示せ．

解答4

● 時間領域[式(89)による]

$$P = \frac{1}{3}\left[5^2 + 5^2 + (-1)^2\right] = 17 \quad \cdots\cdots\cdots\cdots (94)$$

● DFT 値

式(75)～式(77)を式(92)に代入する．

$$P = 3^2 + \left|1 - j\sqrt{3}\right|^2 + \left|1 + j\sqrt{3}\right|^2$$

$$= 3^2 + \left(\sqrt{1^2 + \left(-\sqrt{3}\right)^2}\right)^2 + \left(\sqrt{1^2 + \left(\sqrt{3}\right)^2}\right)^2$$

$$= 9 + 4 + 4 = 17 \quad \cdots\cdots\cdots\cdots (95)$$

● DCT 値

式(78)～式(80)の $\{C_\ell\}_{\ell=3}^{\ell=2}$ を $\{G_\ell\}_{\ell=0}^{\ell=2}$ と読み替えて，式(92)に代入する．

$$P = 3^2 + \left(\sqrt{6}\right)^2 + \left(-\sqrt{2}\right)^2$$

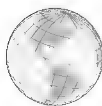
$$= 9 + 6 + 2 = 17 \quad \cdots\cdots\cdots\cdots (96)$$

したがって，式(94)～式(96)はすべて同じ値になり，式(93)が表すパーシバルの定理の正当性が検証される．

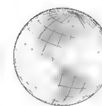
＊　　　＊

次回は，信号数学の総まとめの第 2 弾として“DFT”と“FFT”を採り上げ，その物理的意味を中心に“DCT”と対比させながら，わかりやすく解説する予定にしている．お楽しみに．

みたに・まさあき　東京電機大学工学部情報通信工学科



INFORMATION
海外・国内イベント/セミナー情報



海外イベント

4/26-5/1 **ICRA 2004（IEEE 2004 International Conference on Robotics and Automation）**
Hilton & Towers, New Orleans, LA, USA
IEEE
http://www.icra2004.org/

5/9-14 **NetWorld + Interop 2003 Las Vegas**
Las Vegas Convention Center, Las Vegas, NV, USA
MediaLive International
http://www.interop.com/

5/17-20 **Embedded Processor Forum 2004**
Fairmont Hotel, San Jose, CA, USA
In-STAT/MDR
http://www.mdronline.com/epf04/index.html

5/25-27 **CeBIT america 2004**
Jacob K.Javits Convention Center, New York, NY, USA
Hannover Fairs USA
http://www.cebit-america.com/

6/1-5 **COMPUTEX TAIPEI 2004**
Taipei World Trade Center, Taipei, Taiwan
Taipei Computer Association
http://www.ippc.biz/computex2004/Preregistration.asp

国内イベント

4/27-28 LASER EXPO 2004
パシフィコ横浜（神奈川県横浜市西区）
（社）レーザー学会
http://www.optronics.co.jp/le/

4/27-28 Grid World 2004
東京ファッションタウンTFTホール（東京都江東区有明）
（株）IDGジャパン
http://www.idg.co.jp/expo/grid/

4/29-5/31 『PlayStationと科学』展～コンピュータテクノロジーとエンタテインメントの融合～
日本科学未来館1F 催事ゾーン（東京都江東区青海）
（株）ソニー・コンピュータエンタテインメント
http://www.playstation.jp/cmc/event/index.html

5/1-4 ROBOTREX 2004／ロボカップジャパンオープン2004大阪
インテック大阪（大阪府大阪市住之江区）
ロボカップ国際委員会，ロボカップ日本委員会，ロボカップ大阪大会開催委員会，日本経済新聞社
http://www.nikkei.co.jp/events/robotrex/03.html

5/11-14 ビジネスショウ TOKYO 2004
東京国際展示場（東京都江東区有明）
（社）日本経営協会，東京商工会議所
http://bs.noma.or.jp/

5/19-21 自動車技術展 人とくるまのテクノロジー展2004
パシフィコ横浜（神奈川県横浜市西区）
（社）自動車技術会
http://www.jsae.or.jp/2004expo/index.html

5/19-21 第3回 国際バイオEXPO
東京国際展示場（東京都江東区有明）
リード エグジビション ジャパン（株）
http://web.reedexpo.co.jp/bio/jp/

5/19-22 ビジネス＆テクノロジーフェア2004関西
インテックス大阪（大阪府大阪市住之江区）
日刊工業新聞社
http://www.nikkanosaka.com/event/

5/26-28 2004電設工業展
インテックス大阪（大阪府大阪市住之江区）
（株）日本電設工業協会
http://www.jeca.or.jp/exhibition/

6/3-4 TOPPERSコンファレンス
学士会館（東京都千代田区神田錦町）
NPO法人TOPPERSプロジェクト
http://www.toppers.jp/

開催日，イベント名，開催地，問い合わせ先の順
日程はすべて予定です．問い合わせ先にご確認のうえ，お出かけください．

セミナー情報

スイッチング電源の基礎と実際
開催日時　：5月7日（金）
開催場所　：CQ出版セミナー・ルーム（東京都豊島区巣鴨）
受講料　：13,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎（03）5395-2125，FAX（03）5395-1255

ビギナのための回路定数決定法
開催日時　：5月8日（土）
開催場所　：CQ出版セミナー・ルーム（東京都豊島区巣鴨）
受講料　：3,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎（03）5395-2125，FAX（03）5395-1255

二足歩行ロボットの制御技術
開催日時　：5月9日（日）13:00～17:00 定員30名
開催場所　：岐阜羽島テクノビル2Fセミナールーム（新幹線 岐阜羽島駅前3分）
受講料　：2,000円（研究会会費）
テキスト　：Interface 2004年6月号
問合せ先　：（株）イーエスピー企画 教育マイコン研究会事務局，
☎（058）397-0660，FAX（058）397-0661，E-mail：oficejo@esp.co.jp
http://www.esp.co.jp/calender/index.html

わかる半導体とLSI
開催日時　：5月14日（金）
開催場所　：CQ出版セミナー・ルーム（東京都豊島区巣鴨）
受講料　：16,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎（03）5395-2125，FAX（03）5395-1255

ベーシック・ウェハ・プロセス
開催日時　：5月15日（土）
開催場所　：CQ出版セミナー・ルーム（東京都豊島区巣鴨）
受講料　：16,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎（03）5395-2125，FAX（03）5395-1255

半導体製造装置入門
開催日時　：5月20日（木）
開催場所　：CQ出版セミナー・ルーム（東京都豊島区巣鴨）
受講料　：16,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎（03）5395-2125，FAX（03）5395-1255

半導体パッケージング技術
開催日時　：5月21日（金）
開催場所　：CQ出版セミナー・ルーム（東京都豊島区巣鴨）
受講料　：16,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎（03）5395-2125，FAX（03）5395-1255

二足歩行ロボットの制御技術
開催日時　：5月22日（土）13:00～17:00 定員120名
開催場所　：国立オリンピック記念青少年総合センター（東京都渋谷区代々木）
受講料　：2,000円（研究会会費）
テキスト　：Interface 2004年6月号
問合せ先　：（株）イーエスピー企画 教育マイコン研究会事務局，
☎（058）397-0660，FAX（058）397-0661，E-mail：oficejo@esp.co.jp
http://www.esp.co.jp/calender/index.html

半導体の基板への実装
開催日時　：5月22日（土）
開催場所　：CQ出版セミナー・ルーム（東京都豊島区巣鴨）
受講料　：16,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎（03）5395-2125，FAX（03）5395-1255

インターフェース/デザインウェーブ・テクノロジ・セミナー DSP & FPGAデザイン・ワークショップ
開催日時　：5月25日（火）
開催場所　：パシフィコ横浜 アネックスホール（神奈川県横浜市西区）
受講料　：8,400円（1セッション，税込）
※無料のセッションと展示もあり
※インターフェース/デザインウェーブマガジン年間購読者割引あり
問い合わせ先：CQ出版社TSE事務局，☎（03）5395-1465，FAX（03）5395-3911
http://it.cqpub.co.jp/tse/ifdw200405/

Verilog-HDL入門
開催日時　：5月27日（木）
開催場所　：CQ出版セミナー・ルーム（東京都豊島区巣鴨）
受講料　：13,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎（03）5395-2125，FAX（03）5395-1255

ロボットの作り方～実用化への道～
開催日時　：5月28日（金）
開催場所　：東京工業大学大岡山キャンパス（東京都目黒区大岡山）
受講料　：会員/協賛学会員8,000円，学生4,000円，会員外12,000円（税込）
問い合わせ先：（社）日本ロボット学会 講習会係，☎（03）3812-7594，FAX（03）3812-4628
http://www.sanbi.co.jp/rsj/Info2/Seminar/semi_22201.html

ディジタル変復調の基礎～無線データ伝送からCDMA・UWBまで
開催日時　：5月28日（金）
開催場所　：CQ出版セミナー・ルーム（東京都豊島区巣鴨）
受講料　：13,000円
問い合わせ先：エレクトロニクス・セミナー事務局，☎（03）5395-2125，FAX（03）5395-1255

シニアエンジニアの技術草子 参拾九之段

◆一蓮托生，向こう三軒両隣

旭 征佑

● 住基ネット侵入実験

構築費用805億円，維持費が毎年190億円にもなる新型の大型公共事業，住民基本台帳ネットワーク・システムは，各方面からのセキュリティ問題の指摘でいまだ賑やかなのは周知のとおりだ．以前，長野市で住基ネットの侵入試験が行われたことがある．2月末の最終報告によると，庁内LANなどを経由して住基情報が入った既存の住基サーバや全国の住基ネットにつながるコミュニケーション・サーバなどを攻撃したところ，暗証番号が安易だったり，ソフトウェアの弱点が放置されたままだったため，これらのサーバの管理権限を取得，操作できたという．しかも，総務省が設置した地方自治情報センターは，丸2日半に渡ってこの侵入を検知できなかったという．

これを受けて長野県は，市町村の安全性調査の支援，市町村の住基担当者の研修など，今後の対策も公表した．長野県の審議会が昨年提案した県全体の情報網構築，共同センタの設置といった，住基ネットをより安全に，効率的に運用する四段階の安全対策を，市町村と検討していく方針も示した．

これに対し，総務省サイドは住基ネットそのものから侵入できなかったことから，「侵入実験は失敗だった」といい張っている．ピントがずれていて，あきれてものがいえない．

● 実は人間が信用できない

しかし，まだまだ大きな問題がある．住基ネットはすべてWindowsで組まれている．たとえば，総務省や衆議院議員連盟をはじめとして，あれだけ騒いでいたWindowsのセキュリティの問題はなぜか一切議論に上っていない（詳細は明らかにされていないが，長野市の侵入成功はこの点が原因になっている可能性が高い）．通信回線は一民間業者が用意したVPNを間借りした状態だ．もしNTTが倒産したら，NTTが情報を横流ししたら，…いったいどうなるのか．最近，最大手の超一流企業が国民を欺く（あざむく）例が次々と報告されているではないか．

そして何よりも最大の問題は，住基ネットの人為的な漏洩に対する防御策はどうなっているのかということだ．どんなに技術的に高度なセキュリティで守られていたとしても，そのシステムを構築あるいはメンテナンスする技術者，あるいはデータを直接操作する管理権限をもったオペレータがデータを漏らしてしまったらどうしようもないではないか．

そういえば，最近，顧客情報の漏洩事件が後を絶たない．昨年5月に，ローソンの顧客情報56万人分が漏洩したのを皮切りに，8月には信販大手アプラスで8万人，11月にはファミリーマートで18万人分の漏洩が明るみに出た．今年に入ってからは，2月にYahoo!が450万人，ジャパネットたかたが最高で数十万人，3月にはアッカが201人，東武鉄道が13万人…どうも漏洩の規模も大きくなり，頻繁にもなってきている．十万人程度の個人情報の漏洩には驚かされなくなってきた．東武の記事は，朝日新聞では三面記事として隅のほうにわずか3段組みで報道されただけだ．この原稿が世に出るまでに，また，一つや二つは報道されているに違いない，――といってもまんざら嘘ではないだろう．

これらの事件も，内部犯行とみなされているものが多い．いってみれば，内部犯行こそセキュリティ上の最大の問題なのだ．そしてもっとも悪質で可能性が高いのは，人にやらせるというパターンだという．たとえば，多重債務者から情報を取り出す管理権限をもっている人を探すのだという．その手の犯罪の得意な面々には，他人にデータを盗ませるぐらいのことは，たやすいことかもしれない．ほかにも，脅したり，人質をとったりと，やろうと思えばいろいろな手がある．

● なぜこれほどはびこる

第一に，日本人の団体主義だ．身内の恥をさらすことを良しとしない．そして，会社と自分は一蓮托生，内部のトラブルを暴露するような不届き者は，幸福な人生を保証されない．某大手ハム会社の在庫品のラベル貼り替えを暴露して，ニュースにも登場していた倉庫会社の社長は，たいへん正直に内部の犯行を暴露したものだ．しかし，その後，顧客がみんな逃げてしまい，会社は倒産してしまったという．安定した生活が保証されている公務員には，もっとその傾向が強い．この国民的風土が内部犯行の暴露をしっかりとガードし，内部犯行を行いやすくしているのだ．

二番目に，日本の法整備の問題である．「情報」は，そもそも「もの」ではないので，盗んでも刑法の横領罪にも窃盗罪にもひっかからない．また，不正アクセス禁止法は，その趣旨からしても正当な権限をもつ管理者による情報の持ち出しは規制しない．個人情報保護法も，持ち出した企業は処罰できるが，情



報を持ち出した当人は処罰できないし，公務員は最初から規制の対象外である．他人に対し直接金銭的被害を与えていないので詐欺罪も成立しない．

こうやってみると，法律には抜け目がいっぱいある．つまり，盗んでも当人は一切罪にならないのだから，情報を盗もうと考える人間が出てくるのは当然だろう．まして，他人に盗ませた悪党を罰することもできないのだ．

唯一，罰せられる企業に対する処罰も，実は大したことはない．罰則は最高で6か月以内の懲役，もしくは30万円以内の罰金である．平成14年7月の最高裁判所の判例では，基本個人4情報（氏名，住所，生年月日，性別）の漏洩に対し，1人あたり1万円の賠償金と5千円の訴訟費用を支払うこととなっている．これを基準にすると，企業が100万人の情報漏洩で払うべき金額は，100億円にものぼることになる．

先ほどの例には，顧客情報漏洩の事件では，住所氏名だけでなく，住居の形態，年収などの情報が漏洩しているところもある．また実際に，上記の情報漏洩からメールなどの情報を利用して，高額な通信料を請求した事件も報告されている．これらを考えると，1人あたり1万円程度の賠償ではとても済むはずがない．しかし，実際は企業側から，500円から1000円程度の商品券を送って顧客に謝罪することで済ませてしまうのが一般的になっている．この程度で済むならば，大して対策を行わなくても…．裏を返せば日本企業の危機感のなさ，リスク管理の甘さが，こんなところにも出ているといえる．

● 大量情報を扱う行政や企業の責任は重い

実際には，このように明るみに出るのは，氷山の一角にすぎないはずだ．顧客情報や住基ネットの情報が漏れたかどうかなどというのは，第3者からの通報によって発覚するのが常だが，通報する確率はきわめて低い．その情報に価値を見出しているからだ．さらに，情報そのものが細工され，簡単に情報源がわからぬように細工されるのが常なのだ．

ある業者に確認したことがある．買い取った情報は，できるだけ情報供給元がわからぬように，いくつかのDBにシャッフルしてしまうらしい．業者も，せっかく名簿を買い取ったのに簡単に情報の出元がばれてしまったのでは，商売上がったりなのだという．



大量の個人データを扱う行政や企業なら，当然流出するものとして，セキュリティ対策をどれだけ練ってきたのか，そして漏洩したことに対する責任が問われるはずだ．その意味で過去に漏洩した企業の経営者の他人事のような発言には物足りなさを禁じえない．よくいわれることだが，データ管理者のモラル教育を見直す必要があるだろう．しかし，こんなものは単なる気休めで，効果があがるはずはない．

データを持ち出した人間，管理者，経営者にはもっと重い罪が課せられて良いはずだ．顧客の大事な個人情報を預かっているのだから，万が一にも漏れたら，もっと顧客に大きな損害賠償を払うべきだろう．そして会社をやめた場合，損害賠償がどのくらいになるのかを管理者には知らしめるべきだ．

それから，情報漏洩の真犯人を必ず突き止めるべきだ．情報の盗難者を見つけ出すことが目的のリアルタイム管理システム，ログ管理システムは開発できないのか．情報漏洩をいかに防ぐかだけでなく，情報漏洩が必ずあることを前提にした情報漏洩の犯人を突き止めるシステムの整備を，最新の技術として開発することは急務ではないかと思うのである．

あさひ・しょうすけ　テクニカル・ライター
イラスト　森 祐子

Engineering Life in

フリー・エンジニアという仕事（第三部）

■今回のゲストのプロフィール
ボブ・アイゼンスタッド（Bob Eisenstadt）：LSI設計エンジニアとして過去20年近くの経験を有する．VLSI Technology Inc.（現在はPhilipsの一部）でASIC設計の経験を積んだ後，Supermac, Radius, Silicon Graphics, 3DFX，Silicon Imageなどのグラフィックス関係の企業でLSI設計の外部スペシャリストとして活躍する．そのほかにスタート・アップの設立の経験や，パテント取得の経験を有する．オフには運動，造園や家族と過ごす．マサチューセッツ州ボストン出身．

前回まで

ASICベースの設計からCOTベースの設計にかわりつつあったインターネット・バブル時のシリコンバレーについて話が出た．とくにこの期間でのチップ設計手法についての移り変わりについて伺った．デザイン・チームの大きさとそれぞれのチーム・メンバの役割，そしてシステム全体を取り仕切るアーキテクトの仕事について具体的な経験から得た興味深い話などが出た．

☆ 強烈な個性が多い会社

トニー▷今回は技術的な話からシフトして，いろいろな会社の文化についてお聞かせください．多くの会社でフリー・エンジニアとしてお仕事をされているので，さまざまなスタイルの会社と社員をご覧になったと思うのですが，いかがでしょう？

ボブ☒そうですね，ASICベンダのエンジニアとして働いていたころから，顧客のところに出向いていっしょに仕事をすることがあったので，そこから話しようかと思います．

トニー▷私も同じように顧客のところでいっしょに仕事をした経験があります．大きく分けると，普通の会社，あまりにもリラックスした会社，そして強烈な会社でしょうか？　個人的な経験からするとアップルコンピュータの中央研究所とかNeXTの開発部などが強烈なイメージがあるのですが….

ボブ☒たしかにわが物顔で協力会社のエンジニアとかを平気でいじめたりしますよね（笑）．強烈というのはだいたい人使いが荒いとか，つっけんどんですごく愛想が悪かったりですね．設計チームの全員，そして部長まで強烈で濃いキャラクタな会社があります．現在のアップルはわかりませんが，昔のアップルは多くの著書にも出てくるように，本当に我の強い人達の集まりみたいなところがありましたね．まあ，それだけのこだわりがあって良い製品が出たりするわけですが．

トニー▷また，会社によっては，そういうカルチャを意図的に促進しているような会社もあります．たとえばスタート・アップでよくあるパターンは，「君達は選ばれた人達だ！　君達が世界を変えて行くのだ！」みたいに社員達に自分達は特別賢い人間の集団である…みたいな感覚を植えつけるような．まあ，給料が低いし勤務時間は長いのが平気なスタート・アップだから，こういい続けることによって気合いを入れるという面もありますね．

ボブ☒スタート・アップではそういうことが必要と思うのかもしれません．ちょっと洗脳ぽいですよね．でも，それぐらいしないと社員がついてこないと思っているのかも．アメリカ人は日本の会社でラジオ体操をしたり社歌を歌うのを笑っているけれど，我々シリコンバレーでも「我々はナンバー・ワンだ！」とかいっしょに叫んで士気を上げるお祭り騒ぎをしているのだから，他人のことを笑ってられませんね（苦笑）．

トニー▷おっしゃるとおりです．そのほかでよくある社風は，スター・エンジニアを育てて，社内競争を激しくすることによって良い仕事を期待しているとか？

ボブ☒うんうん，それはありますね．私が最近経験した会社の中では，NVIDIAとかはそのタイプだと思います．とにかくできるエンジニアをスターにしていくようです．できるエンジニアを学会に出して論文発表などさせて，社内でも表彰するとかで社内を盛り上げようとしていますね．

しかし，逆もあってチームワークを重視する会社もあります．仕事の内容によっては，スター制のほうが良いケースもあるし，そうでない場合もあります．だからこんなにたくさん会社があって，賢いエンジニアが大勢いても成功する会社と失敗する会社があるのだと思います．

トニー▷でも，個人レベルで見てどうでしょう？　エゴの塊みたいなエンジニアとかいますよね．大学卒業したてのエンジニアで，愛車のナンバ・プレートが「VLSI DSGN」で，プレートのホルダに“World's Best Engineer”（訳：“世界一のエンジニア”）とか平気で入れて走っている人とか過去いましたよ．

ボブ☒いましたねぇ…そういう若いエンジニアが（苦笑）．仕事にプライドをもっていること自体は悪くないと思いますが，このエンジニアのように，自分の存在感と職業を直結している人達がいるのだと思います．まあ，仕事があって，会社の資金も潤沢のときは，ことがだいたいうまく運ぶけれど，景気が悪化したり，会社の状態が悪化すると一気に変わりますよね．

トニー▷そうですか…．私は性格的なものだと思っていましたが．

ボブ☒元はそうだと思います．でも，今回のように長い不況が続いたりレイオフされたりして，一皮剥けていくのだと思います．つまり，周りの人達とうまく仕事をしたり，レイオフされても仲間がいれば転職先とか見つかりやすいとわかるようになるのだと思います．自分のエゴだけではどうにもならないですから．後はレイオフされたりして，やっと自分を見つめて仕事以外の自分に気が付くとか．

☆ 仕事を確保していく努力

トニー▷今のところ，まだシリコンバレーは景気が悪いのですが，レイオフとかに対してエンジニアとしてできることはありますか？　また，レイオフの予防法などはあるでしょうか？

ボブ☒レイオフは会社のつごうなどもありますからね．でも自分の価値を上げるということでは，いろいろ考えられると思

います．まずは，(1)自分のネットワークを持つことによって情報源を拡大することができます．(2)スキルアップなど日々心がけることによって新しいスキル，たとえばこれから主流になるプログラミング言語を覚えるなどすることです．また何を知っていれば仕事が増えるかも知っておく必要があります．最後に，(3)柔軟に対応できるようにしておくことです．これは，たとえばレイオフされた後に，必ずしも自分の希望している職につけるわけでもないし，あんまり職種にこだわりすぎるとチャンスを見逃すかもしれません．

トニー これらは，レイオフされた後の対処法ですね．ボブさんの場合，自分が資本だし売り物だからやっぱりこういうスキルをちゃんと身に付けていますね．

ボブ 今後はエンジニアのスキルがピークに達する時期が低年齢化してくると感じます．つまり，昔は50代ぐらいまで続けられたのが，今はせいぜい40代ぐらいまでですよね．新しい技術が出てくるし，最近ではグローバル化もあるのでエンジニアとして旬なときが過ぎていくのが早く感じます．だから必死で新しいことを覚える必要があると思うのです．

トニー 年齢的な限界は自分の努力で何とかなる気もしますが，雇用側や会社側が年齢差別をしているとは思いませんか？

ボブ アメリカでは年齢制限は明らかに違法です．しかし，チップ設計やプログラミングのエンジニアのことを考えると，雇用側の理想は大卒で5～6年実務経験のある30代前半の男性エンジニアでしょうね．まだ独身か子供がいないので長時間の勤務も問題ないとか…計算していると思います．だから，レジュメでは年齢を書いたりしませんが，卒業の時期からだいたい割り出しているので，弾かれてしまう候補とかいますよね．

トニー 私も知り合いの人材斡旋の方から聞いたのですが，6か月以上求職しているエンジニアは自動的に「難しいケース」と見なされてしまい，断られたりするらしいですね．

ボブ それはキツイですね．最近だと早く出世してしまって，管理職とか営業とかで，現場から離れる若手のエンジニアが増えてますよね．とくにインターネット・バブル時は社会に出たての若手エンジニアが起業家になったり社長や役員になってました．バブルが弾けた後，技術職でない方向に進んだ人は苦労したと思います．エンジニアとして復帰できるスキルのある人はよかったと思いますが，管理職とか営業・マーケティング関係にいった若手エンジニア達はたいへんだったと思います．管理職とか営業・マーケティングの仕事って一番最初にカットされるポストですからね．

トニー いろいろな経験をするのは良いと思いますが，エンジニアとしての旬とか，新しいことを覚える努力はいつになっても大切です．また，全体的に自分のキャリア・プランを考えながらバランスよく仕事をすることも大切です．先ほどのエゴの塊みたいなエンジニアのように，仕事以外の自分がないとレイオフされたときとか精神的にキツイでしょうね．

☆ 今後のシリコンバレーに期待を持てるか？

トニー 今後シリコンバレーはどういうところになっていくと思いますか？ また，何が気になりますか？

ボブ 私はグローバル化とかはあまり気になってません．インドとか低賃金の国々にエンジニアの仕事が流れるといった心配です．ある種の手離れの良い仕事，つまり仕様書できっちり説明できるような仕事は流れていくと思いますが，作り込みが必要で多数のエンジニアや外部の顧客のフィードバックが必要な複雑なプロジェクトはアウトソーシングしにくいと思うからです．

トニー 分野的には何が気になりますか？

ボブ 今のところ一番気になっているのが，シリコンバレーが最近の流行物のリーダーシップをあまり取っていないところでしょうか．LCDモニタ，プラズマ・テレビ，HDTVなどの家電がアメリカではほとんどないし，携帯電話や無線通信もあまりシリコンバレーがリーダーシップを取っているといえません．

トニー コンピューティングが中心でその中に入るチップ，ハードディスク類とか，ネットワーク機器類が強いですよね．

ボブ そう，どちらかというと本当にオーソドックスなコンピューティングが中心ですね．だからIntel，HP，Sunがどうしてもシリコンバレーを代表するような会社になりますね．Qualcommとかはサンディエゴだしね．

トニー HPのCEOフィオリーナ氏も最近証券アナリストに対して今後は二桁台の成長率は見込めず，成熟した業界だといっていました．

ボブ 成熟した業界だと困りますね．昔は，メインフレームとかオフコンを使っていた時代にアップルがApple IIを紹介することによってパーソナル・コンピュータができたわけですが，現在のやりかたや考えかたをひっくり返すような画期的なことをやらないと確かに成熟した業界ですね．しかし，シリコンバレーのおもしろいところはこんなに狭い場所にエンジニアの密度が高いのでアイデアが出やすいし，アイデアを実現に向けて動ける人達の密度が高いと思うのです．ですから新しいことが起きると期待は持っています．

対談を終えて：

ボブさんは，筆者が社会に出たての頃にお会いしたエンジニアだ．アメリカ人にはまれな謙遜するエンジニアだ．今回はいろいろと本音でシリコンバレーでの仕事の辛いことや，シリコンバレー企業の裏舞台のような，本音で話が聞けたのがとても新鮮だった．

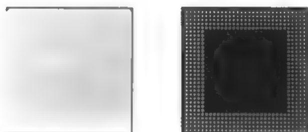
トニー・チン htchin@attglobal.net WinHawk Consulting

●車載情報端末用システム・オン・チップ

SH7770

- 720MIPSの高処理性能を実現したCPUコア「SH-4A」を搭載し，地図描画用2Dグラフィックス・エンジン，多彩な描画処理が可能な3Dグラフィックス・エンジンを内蔵．
- 「SH-4A」は，最大動作周波数が400MHzで，キャッシュ・メモリを4ウェイ化してキャッシュのヒット率を向上させている．
- 2Dグラフィックス・エンジンの太線描画機能は，道路を描画する際，方向に依存せず，一定の線幅で描画することなどが可能．
- アンチエイリアス機能により，多角形や斜め線などの滑らかな描画を実現．
- 3Dグラフィックス・エンジンとして，英国イマジネーション・テクノロジー社の3DグラフィックスIPである「PowerVR MBX」を採用．

●サンプル価格：¥8,000



■（株）ルネサステクノロジ
TEL：03-5201-5120

●少ピン，小型パッケージ・マイコン

R8C/14，R8C/15 R8C/16，R8C/17

- 20ピンでありながら，高性能，高機能を実現したフラッシュ・メモリ内蔵マイコン．
- 入出力ポート13本，入力ポート2本の最大15本のI/Oポートの使用が可能．
- 電源電圧3V以上で，動作周波数は最大20MHzの高速動作であり，周辺機能はアウトプット・コンペア機能やパワーオン・リセット機能，電圧検出回路を搭載．
- チップ・セレクト付き同期式シリアルI/O機能（R8C/14，R8C/15シリーズ），I²Cバス・インターフェース（R8C/16，R8C/17シリーズ）を搭載．
- 標準が8MHzの高速リング発振器を搭載し，内部レジスタをプログラムで書き換えることにより，周波数調整ができ，補正回路を内蔵したことで，±3%以下の周波数精度を実現．

●サンプル価格：¥250～¥330



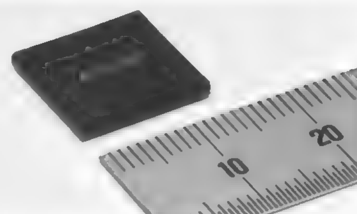
■（株）ルネサステクノロジ
TEL：03-5201-5219

●10GビットEthernet用トランシーバLSI

M69850AWG

- 送受信に関わるPMA，PCS，XGXSの三つの機能ブロックをワンチップ化．
- 10GビットEthernet用シングルチップ・トランシーバICとして，10GビットEthernet規格（IEEE802.3ae），10Gビット・ファイバ・チャネル規格（T11.2 10GFC），XENPAK-MSA（Revision3.0）規格に対応．
- SOI-CMOSは，チャネル層と基板層がシリコン酸化膜で物理的に絶縁された構造なので，トランジスタの寄生容量が少なく，トランジスタ動作速度が高速化されるとともに，消費電力を低減．
- 10Gbpsの高速動作を，1.2Wの低消費電力で実現．

●サンプル価格：¥35,000



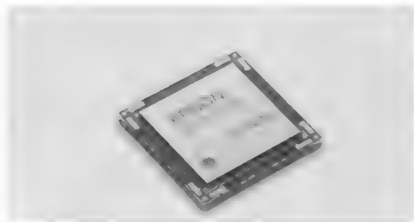
■三菱電機（株）
TEL：03-3218-4772 FAX：03-3218-4862

●動画の録画・再生機能付き表示コントロールLSI

S1D13732

- 動画の録画，再生機能をサポートし，メガピクセル・カメラを搭載する携帯電話機やPDAなどに適する．
- MPEG-4および，欧州で主流となっている動画方式H.263機能をハードウェアで搭載．
- CPUで動画のソフト処理を行わずに，動画の録画/再生が可能になり，携帯電話の低消費電力化を実現．
- カメラ・インターフェースは，最大SXGA（約131万画素），1系統を搭載．
- ハードウェアYUV/RGBコンバータを内蔵．
- ホスト・インターフェースは，16ビットの80系/68系CPU IndirectおよびDirectインターフェース，シリアル・インターフェースをサポート．

●サンプル価格：¥2,000



■セイコーエプソン（株）
TEL：042-587-7665

●低価格CPLD

CPLD MAX II

- ルック・アップ・テーブル・アーキテクチャを採用し，TSMC社の0.18μmフラッシュ・プロセスによって製造．
- 従来品と比較して，1/2のコストと1/10の消費電力を実現．
- インスタント・オン，ワン・チップ・ソリューション，不揮発性および優れた操作性などの特徴を備える．
- 配線構造，ソフトウェア・アルゴリズム，プロセス技術の改善により，性能および集積度が向上．
- ユーザ・フラッシュ・メモリの搭載や，リアルタイムISPなどの機能を搭載．
- CPLDデザイン向けのデザイン・ツールである，アルテラQuartus IIバージョン4.0デザイン・ソフトウェアを利用することで，デバイスの設計開発を開始できる．

●価格：$1.50～$7.00（500,000個時）

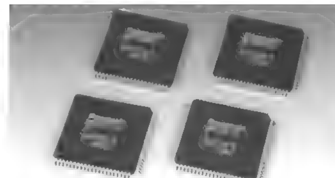
■日本アルテラ（株）
TEL：03-3340-9480 FAX：03-3340-9487

●アナログ放送テレビ用システムLSI

ALGISTA8シリーズ

- アナログで行っていた処理を，ディジタル信号処理で実現しており，回路のバラツキがなくセット生産での画質調整が不要．
- 周辺部品をディジタル信号処理回路に組み込み，外付け部品点数の大幅な削減が可能．
- PAL，SECAM，NTSC，文字放送という世界の放送方式に合わせた4種類のLSIを入れ替えることで仕向け地を変更することが可能．
- 4096種類の色表現に加え，16通りの色の付いた半透明色を表示することのできるOSD表示回路と，アジア欧州圏で行われているテレテキスト方送受信回路，北米圏で行われているクローズド・キャプション受信回路により，世界の文字放送に対応．

●価格：下記へ問い合わせ



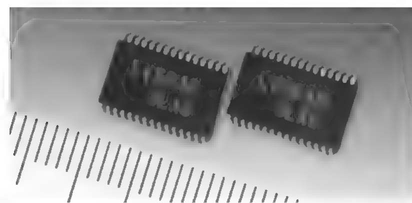
■松下電器産業（株）
TEL：075-951-8151
E-mail：semiconpress@scd.mei.co.jp

●ディジタルAV機器用DC-DCコンバータ用制御IC

LV5040V

- 一つのチップ内に二つの同期整流回路を持ち，それぞれの整流回路のスイッチング位相を逆にして動作させるため，入力側のピーク電流を抑えることで，平滑フィルタを小さくすることが可能.
- 外部部品構成を変えるだけで，容易に一出力二相運転を実現.
- クロック出力端子を持ち，外部クロックに同期してスイッチング動作することが可能.
- 複数デバイスを使用した場合に，発生しやすい電圧波形の乱れやビートの発生を抑制することが可能.
- 電源停止時に出力に現れるリンギングを外付けの放電トランジスタなしで制御可能な機能を搭載.

●サンプル価格：¥250



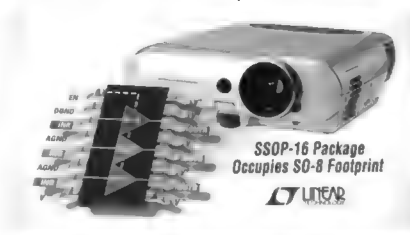
■三洋電機（株）
TEL：0276-61-8107 FAX：0276-61-8730

●UXGAディスプレイ対応RGBビデオ・アンプ

LT6553

- 外付けの利得設定抵抗が不要で，650MHzの－3dB帯域幅を特徴としている.
- 2500V/μsの高スルーレートと6nsの短いセトリング・タイムにより，AC特性が向上し，よりシャープなビデオ画像を可能にする.
- 0.01°の微分位相と0.01%の微分利得により，優れた直線性が得られるため，シグナル・インテグリティを維持できる.
- 0.1dBの利得平坦性を150MHzまで維持するため，広範なビデオ信号で使用しやすくなっている.
- イネーブル/ディセーブル機能を搭載しているので，マルチプレクスや信号経路指定が容易.

●サンプル価格：¥270（1,000個時）



■リニアテクノロジー（株）
TEL：03-5226-7291 FAX：03-5226-0268

●10GビットEthernet向けトランシーバー

TLK3118

- 10GビットEthernet向けIEEE802.a/eに準拠し，高い柔軟性を持つ，冗長XAUIシリアル・ドライバで，130nmCMOS技術で製造されている.
- LAN，WAN，MANなどさまざまな規模のネットワーク・アプリケーションに最適化されており，基板面積および設計時間の削減を実現.
- 基板面積，システム・コスト，消費電力の削減，小面積により多くの高速チャネル/ポートの集積化を実現.
- デバイスは，物理層インターフェースにおけるパラレル-シリアル変換，シリアル-パラレル変換，クロック抽出機能を提供.
- 差動カレント・モード・ロジックを使用する高速シリアル・トランスミッタは，プル・アップ抵抗を内蔵.
- 送受信イコライザ技術を駆使し，FR4バックプレーンをはじめとする業界標準メディアに最良の信号の完全性を実現.

●サンプル価格：$60（1,000個時）

■日本テキサス・インスツルメンツ（株）
FAX：0120-81-0036

●コントローラ・チップ

CY7C53120L8
CY7C53150L

- LonWorksの制御で使用されるセンサとアクチュエータ，および自動化アプリケーションをネットワークで結び，インターネットに接続することが可能.
- 5Vのピン互換製品と比較して，最大2倍のメモリを利用でき，消費電力は33%低減される.
- シリアル通信インターフェース，2.5Mbpsまで拡張可能なシリアル周辺装置インターフェース・エンジン，プログラム変更可能な低電圧インヒビタを搭載.
- アプリケーション・コードとネットワーク・トラフィックを最大20MHzの速度で同時処理できる3個の8ビットのパイプライン型プロセッサを装備.
- CY7C53120L8は，16KバイトのROM，4KバイトのRAM，8Kバイトの内部EEPROMを装備.
- CY7C53150Lは，4KバイトのRAMと2.7KバイトのEEPROMが組み込まれており，56Kバイトまでアドレッシング可能な外部メモリ・インターフェースを使用.

●価格：CY7C53120L8 $3.80（10,000個時）
CY7C53150L　$3.10（10,000個時）

■日本サイプレス（株）
TEL：03-5371-1921 FAX：03-5371-1955

●低ドロップアウト・リニアレギュレータ

IRU1502-33

- 最大負荷電流が1Aで内部電圧降下が0.6V以下で，スイッチにPチャネルMOSFETを使用.
- 超低雑音，高速スタートアップ，および電源ラインと負荷の過度応答が1A/μsなどを特徴としている.
- 出力電圧は3.3V固定で，熱抵抗は42℃/W.
- 電池駆動の携帯機器，ハード・ディスク装置，CD-ROM装置，DVD装置，ADSLモデムなどの用途に適する.
- 外付けコンデンサとして等化直列抵抗が低いセラミック・コンデンサを使用できることで，低雑音で負荷の過渡応答特性を改善できる.
- 入力電圧が4.75V～5.25Vの範囲，かつ温度が0～120℃の範囲で，出力電圧は標準で3.3V.
- 出力電流が2mA～1Aの範囲で，静止時の消費電流は650μA.
- 標準1.4Aの電流制限機能と，135℃でシャットダウンする温度保護機能を内蔵.

●サンプル価格：¥90

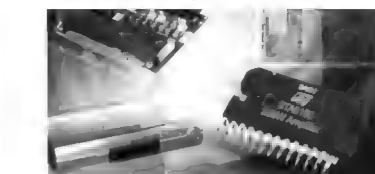
■インターナショナル レクティファイアージャパン（株）
TEL：03-3983-0086 FAX：03-3983-0642

●オーディオ・パワーアンプ

STA5150

- Indigo ManufacturingのBASH高効率トポロジと，Bipolar-CMOS-DMOSプロセス技術を組み合わせた，マルチチャネル・アプリケーションをターゲットとする，オーディオ・パワー・アンプ用チップ.
- BASH技術は，AB級モードで動作するDMOSパワー・トランジスタを使用するとともに，D級スイッチング・アンプのようにひずみを減少させる.
- ひずみ10%未満の4Ω負荷の条件で，従来のAB級と比較して3倍の効率で，200Wを供給.
- 200Wのオーディオ・パワー・アンプ用チャネルを一つ装備し，A/Vレシーバで使用される5.1チャネル・サラウンド・サウンドなどのマルチチャネル・マルチメディア・オーディオ・システムに使用できる.

●サンプル価格：¥1,000



■STマイクロエレクトロニクス（株）
TEL：03-5783-8260 FAX：03-5783-8216

■弊誌では新製品に関するニュース・リリースを募集しております.
宛先は，〒170-8461 東京都豊島区巣鴨1-14-2 Interface編集部ニュース・リリース係
FAX：(03)5395-2127，E-mail：mngnews@cqpub.co.jp　（編集部）

●磁歪式ポジション・センサ

ポジクロン PCFCシリーズ

• 非接触型の絶対位置センサで，耐摩耗性に優れた位置測長システム．
• ロッド型，プロフィール型および超平坦プロフィール型など多種の形状があるため，センサのあらゆる設置状況に合致．
• 位置決定のための可動磁石と，磁歪波伝播用ガイドから構成される．
• センサの測定原理は，Wiedemann効果とVillary効果の二つの物理効果に基づいている．
• 射出成形機，供給混合システム，鋳造機器，車載制御機器，トンネル掘削設備機器，風力発電プラントなどのアプリケーションに適している．
• IP68までの保護等級をサポートし，耐振動衝撃性に優れている．

●価格：下記へ問い合わせ



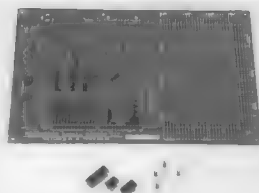
■サンシステムサプライ（株）
TEL：03-3397-5241 FAX：03-3399-2245

●8ビット・マイコン評価ボード

MC68HC908QT/QYシリーズ用体験デモキット

• 基板には，加速度，圧力，人感センサ入力，ソレノイド出力，モータ出力，オープン・コレクタ出力を備え，それぞれがスイッチを実装することで切り替えが可能．
• ユニバーサル領域を大きく取ることで，多種多様なアプリケーションの評価に適するように設計されている．
• キット上のマイコンにプログラムを書き込むには，統合開発ソフト「CodeWarrior Development Studio for HC08 Special Edition」上でソフト・シミュレーションを行った後，パソコンからシリアル・ケーブルを介して行う．
• 評価ボード上でデバッグが必要な場合は，Cyclon-Pro（別売）を使用することが可能．

●価格：下記へ問い合わせ



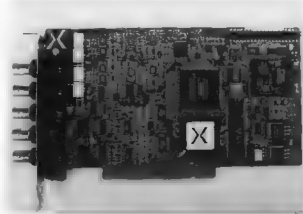
■丸文（株）
TEL：03-3639-8951 FAX：03-5645-5330

●データ収集制御ボード

model CHシリーズ

• ホストPCのRAMサイズによって制限される記録長と，非リアルタイムOSであってもアナログ信号の連続的収集または再生機能を供給するにあたり，16Mバイトのオンボード・メモリ，ローカル・プロセッサ，およびPCIバス・マスタリング機能を最大限に利用．
• 2チャネル入力，2チャネル出力のダイナミック特性で，IQ変調と復調などコミュニケーション関連アプリケーションに適する．
• 12ビット分解能の高速精度でフレキシブルなトリガ機能を装備．
• オンボードのDSPコプロセッサは，高レベルのアルゴリズムとアプリケーション処理を担当．

●価格：¥214,000～¥270,000



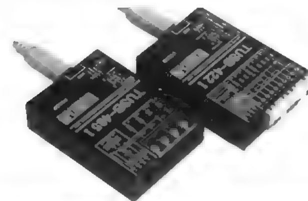
■サンシステムサプライ（株）
TEL：03-3397-5241 FAX：03-3399-2245

●USB-RS-422/485変換ユニット

TUSB-422I TUSB-485I

• パソコンの標準インタフェースであるUSBから，RS-422またはRS-485規格に変換するコンバータ．
• USBの利便性とRS-422/485の長距離伝送機能，マルチドロップ対応機能をあわせ持ったデータ通信システムの構築が可能．
• インターフェースは，USB1.1に対応．
• 通信部にIC内蔵フォトカプラ，電源部にIC内蔵DC-DCコンバータを搭載．
• 絶縁耐圧は1分間にAC500Vで，絶縁抵抗はDC500Vで100MΩ以上．
• 最大転送速度は250Kbpsで，最大転送距離は総延長1.2km．
• 消費電流は，約80mA．
• 接続ユニット数は，最大で32台．

●価格：各¥19,000



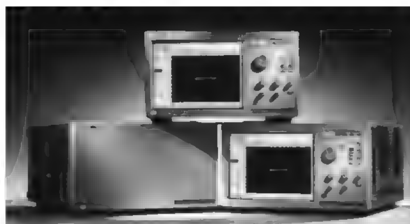
■（株）タートル工業
TEL：029-843-0045 FAX：029-843-2024

●ロジック・アナライザ

16900シリーズ ロジック解析システム

• 3スロットのメインフレームをサポートすることにより，102チャネルのカードと併用した場合，従来品と比較して約30%の低価格でロジック・アナライザの導入が可能．
• Windows XP Professionalベースのオープン・プラットホームを採用することで，オフライン解析やプラットホームの複数構成などを行うことができる．
• FPGA内の信号モニタをロジック・アナライザから行うことができる「Virtual FPGA Probing」機能を搭載．FPGAをリコンパイルすることなく，FPGA内の最大8192か所から最大256の信号グループをロジック・アナライザから指定して測定，表示させることができる．

●価格：約¥2,590,000～



■アジレント・テクノロジー（株）
TEL：0120-421-345

●ハードディスク用プリアンプ

HDL6D300シリーズ

• 従来品と比較して，2倍の遮断周波数を持つ「SOI SiGa BiCMOSデバイス」を採用．
• 書き込み速度と読み出し帯域を向上させ，2.5Gbpsの内部データ転送速度を実現．
• 幅広い読み出しヘッド抵抗値に対応するため，新開発の「並列負帰還読み出し回路」を採用．
• 読み出しヘッド特性に合わせてプリアンプを再度設計する必要がなくなるため，高性能サーバやストレージ・システム用HDDの開発期間を短縮することが可能．
• 独自の「ゼロ・コモン書き込み回路」の回路方式を最適化することで，高精度化を実現し，磁気ヘッド・ディスク間の放電や集塵を防止したほか，書き込み信号の読み出しヘッドへのクロストークを大幅に低減し，読み出しヘッドの保護を実現．

●サンプル価格：¥3,000

■（株）日立製作所
TEL：0428-33-2011

●レベル2キャッシュ・コントローラ

L210コントローラ

- ARMプロセッサと組み合わせることで，25〜75%の性能向上，バッテリの長寿命化およびメモリ・コストの低減を実現．
- モトローラ社のMXCアーキテクチャに搭載することにより，端末のバッテリ消費を増やすことなく，性能向上が可能．
- CPUのオフチップ・メモリに対するメモリ・アクセス数が減少するため，ARM1136JF-SコアなどのCPUに必要な外部メモリ・アクセスに使用する帯域を小さく抑えられる．
- 統合された命令/データ・キャッシュを採用し，128Kバイト〜2Mバイトのレベル2キャッシュに対応．
- ARM1136JF-Sコア，ARM1136J-Sコア，ARM1026EJ-Sコア，ARM926EJ-Sコアに対応．

●価格：下記へ問い合わせ

■アーム（株）
TEL：045-477-5260 FAX：045-477-5261

●低電圧DC-DCコントローラ

TPS40020/TPS40021

- 最高92%の電力変換効率を実現する一方，工業製品，通信機器，データ通信機器，ワイヤレス通信基地局，各種サーバ・アプリケーションにおいて，非絶縁型電源システムの基板面積の削減，回路設計の簡素化を実現．
- ハイ・サイドおよび同期整流回路のNチャネルMOSFETに関連するダイオード導通損失を最小に抑える．
- 同社の「Predictive Gate Drive」テクノロジと内蔵チャージ・ポンプ回路を組み合わせて，安定化5Vゲート・ドライブを実現．
- ソフト・スタート，ハイ・サイド検知の短絡保護などのプログラマブルの電源機能を提供し，より広範囲の動作を実現．
- 100kHz〜1MHzまでのプログラマブルな固定周波数で動作し，迅速な応答速度を提供するパワー・グッド信号出力付き出力電圧トランジェット・コンパレータを搭載．

●サンプル価格：$1.15（1,000個時）

■日本テキサス・インスツルメンツ（株）
TEL：0120-81-0036

●ネットワーク・プロセッサ

Au1550プロセッサ

- セキュリティ・エンジンとして，IPSecとSSL VPNスタンダード双方をサポート．
- すべてのVPNパケット処理をハードウェア上で実行するため，高速化を実現．
- DES，3DES，AES，ARC-4，SHA-1，MD5を実装．
- True Random Number Generationを内蔵．
- DDRおよびSDRAMをサポート．
- 無制限の複数VPNトンネルを，同時に保持．
- SafeNetからライセンスされた知的財産テクノロジ，SafeXcel IPによって実現され，ネットワーク機器用の卓越したセキュリティ性能の提供を可能にしている．
- OSはWindows CE，.NET，Linux，VxWORKSに対応．

●サンプル価格：
333MHz版 ¥2,420（10,000個時）
400MHz版 ¥2,970（10,000個時）
500MHz版 ¥3,740（10,000個時）

■日本AMD（株）
URL：http://www.amd.com/jp-ja/

●10GビットEthernetレイア2スイッチ

XGシリーズ

- 1チップ10GビットEthernetスイッチLSIにより，12ポートの10GビットEthernetをフルワイヤ・スピードでスイッチングできる，240Gbpsの高スループット性能を実現．
- 450nsの低遅延性能を実現．
- 大容量の高速データ転送を要求される，IDC，ISP，IX，HPC市場に適する．
- 8個の10GビットEthernetポートを内蔵しながら，高さ2Uの小型筐体を実現．
- 10GBase-CX4対応タイプでは，光モジュールによる光ファイバ伝送の代わりに，銅線ケーブルによる10Gビット信号の25m伝送を実現．
- IDCなどを構築する場合に，接続距離に応じて最適なインターフェースを選択できる．

●価格：¥8,000,000〜

■（株）PFU
TEL：044-540-4536

●多チャネル温度測定用データ・ロガー

Midi Logger GLシリーズ

- 多チャネルで電圧，温度，パルス（電力積算，回転数，流量積算）のマルチ・ファンクション対応測定器．
- A5サイズで，各種入力信号によるチャネル間の影響を受けない，全チャネル絶縁入力方式を採用．
- 入力部は，10/20/50チャネルの3種類の端子台が，いずれか2セット，ワンタッチで着脱でき，測定点数に合わせ端子台を組み合わせれば，最大100チャネルの計測が可能．
- 測定点が増えた場合には，端子台だけでチャネルの増設が可能．
- 複数の測定対象に端子台のみ先に取り付けておくことも可能．

●価格：¥220,000（GL400）
¥170,000（GL350）



■グラフテック（株）
TEL：0120-626294

●ディジタル・アップ・コンバータ

AD6633

- クレスト・ファクタ・リダクション技術を備えた，CDMA2000，W-CDMA，TD-SCDMAなどの3Gワイヤレス送信機アプリケーション向けの製品．
- 125Mspsの速度で動作し，4または6チャネルを処理し，クレスト・ファクタの低減と信号ひずみ調整を可能とする．
- 信号ひずみは個別チャネルにダイナミックに配分できるため，高品質なデータ通信か，低品質な音声通信かといった性能の優先順位の設定が可能．
- プログラマブル・ワイドバンド・チャネル・フィルタ機能を搭載．
- 6個の処理チャネルで共有される20ビットの入力ポートと，18ビットのパラレル出力ポートをもつことが特徴．
- 全域通過位相等化フィルタ，再サンプリング付きプログラマブルRAM係数FIRフィルタなどを内蔵．

●サンプル価格：
6チャネル版 $60.00（10,000個時）
4チャネル版 $40.00（10,000個時）

■アナログ・デバイセズ（株）
TEL：03-5402-8291

■弊誌では新製品に関するニュース・リリースを募集しております．
宛先は，〒170-8461 東京都豊島区巣鴨1-14-2 Interface編集部ニュース・リリース係
FAX：（03）5395-2127，E-mail：mngnews@cqpub.co.jp （編集部）

NEWPRODUCTS

●ダイナミック・リコンフィギュラブル・プロセッサ

DAP/DNA-2

- 専用演算器を多数内蔵し，アプリケーションに応じて最適な回路を構成できるプロセッサ．
- 専用演算器の構成は，システム構築時だけではなく，システムの動作中にもアプリケーションに合わせて再構築できる．
- 専用演算器を2次元に配列することによる高速性とダイナミック・リコンフィギュラブル・プロセッサ技術による柔軟性を兼ね備える．
- 従来，複数チップを必要としていた機能を，ワンチップで実現．
- 統合開発環境である「DAP/DNA-FW II」を使用することで，高級言語から直接，チップの開発を行うことができる．
- 「DAP/DNA-FW II」は，アプリケーション開発において，アルゴリズム・デザインから実デバイス上での検証までの全開発プロセスをカバーしているツール・セット．
- DNA Designer機能を加え，グラフィカルにアルゴリズムを構築することが可能．

●価格：下記へ問い合わせ

■アイピーフレックス（株）
TEL：03-5436-3861 FAX：03-5436-3862

●ビデオ再生ソフトウェア

H.264/MPEG-4 AVC デコード・ソフトウェア

- 携帯電話内部で使用されている約150MHzの32ビットCPUと16ビット・バスで，8MHz動作の外部メモリ使用の条件下で，QCIF（176×144ピクセル）サイズで15～40fpsの動画コンテンツの再生を実現．
- 使用するCPUやメモリの動作速度を高速化すれば，対応画像サイズやフレーム・レートをQVGA（320×240ピクセル）やVGA（640×480ピクセル），30fpsなどに上げることが可能．
- 因数分解を活用し，計算を簡略化することで，演算回数を大幅に削減．
- 折り返し理論を活用して，数値空間を狭くして，演算回数を削減．
- 演算式を階層化し，優先度を付けて必要な計算だけを実行．
- 複雑な計算式を，階層化することによって簡略化．
- 関数を効率的に変換．

●価格：下記へ問い合わせ

■（株）テクノマセマティカル
TEL：03-5783-2681 FAX：03-3474-5846

●PCB設計向けプラットホーム

Allegro system interconnect design platform

- 設計ツールと解析ツールを使用して，ICからパッケージ，PCB設計までの異なる設計領域のコデザインを可能にし，設計の繰り返しを最低限に抑えることが特徴．
- システム設計のデザイン・チューン全体を通じて，協調的な設計環境の提供が可能．
- 検討，設計，検証，製造などのそれぞれの段階で，フィードバックによる精度合わせを行い，工程ごとにスペックからシステム・インターコネクトを展開．
- 共通のコンストレイント・ドリブン・フローを，デザイン・エントリの段階から，シグナル・インテグリティ，パワー・インテグリティを含む，システム・インターコネクトを統合的に実現．
- シリコン・デザイン・イン・キットを使用した新しい手法を導入し，PCI Expressに対応したデザイン・イン・キットをあわせてリリース．

●価格：下記へ問い合わせ

■日本ケイデンス・デザイン・システムズ社
TEL：045-475-2311 FAX：045-471-7772

●リアルタイム・デバッグ・ソリューション

ChipScope Pro 6.2i

- 再プログラムが可能という特徴を利用して，論理回路のロジック・デバッグ，および組み込みシステムのデバッグにフレキシブルに対応し，設計工数を削減．
- アジレント・テクノロジ社の第2世代のTrace Core（ATC2）を導入することで，同社のロジック・アナライザとの直接リンクを可能にしている．
- デザイン作業の途中でFPGAの中にATC2を挿入したり，論理合成後の回路ネットリストにコアを挿入することが可能．
- FPGAの中に組み込まれたATC2は，任意のFPGA内部信号に対してロジック解析を行うための信号測定アクセス手段を提供．
- PC上で，デザイン変更や再コンパイルを行わず，かつ回路のタイミングを変更せずに，観測する信号を簡単かつダイナミックに変えることが可能．
- FPGAの信号名は新しい測定が行われるたびにロジック・アナライザの中で自動的に更新され，システム検証の際に時間のかかるステップを自動化．

●価格：下記へ問い合わせ

■ザイリンクス（株）
TEL：03-5321-7740 FAX：03-5321-7762

●携帯電話向けトータル・ソリューション

NetFront Mobile Client Suite

- モジュラ構造による柔軟で拡張性の高いトータル・ソリューションで，高度な携帯電話機能を効率的に開発することが可能．
- ブラウザ，カスタマイズ可能なユーザ・インターフェース，MMS Client，PIM，SyncML，JavaVMなどを提供．
- OSに依存しない構造となっており，Agere，Broadcom，EMP，Infineon，SH-MobileやTTPComなどの組み込みモバイル・プラットホームへの移植が可能．
- NetFront Message Clientは，リソースの限られたハードウェア環境において，MMS，M-IMAP，SMTP/POPのメーラ・タイプに対応するメール・クライアント．
- NetFront PIMは，アドレス帳，カレンダ，スケジュール表，To-Doリスト，メモ帳の体系化を実現．
- NetFront SyncMLは，OMA SyncML標準規格v1.1.1に準拠した，クライアント・ライブラリ．

●価格：下記へ問い合わせ

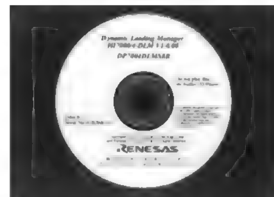
■（株）ACCESS
TEL：03-5259-3685 FAX：03-3233-0222

●μITRON用ミドルウェア

ダイナミック・ローディング・マネージャ

- 携帯情報端末の組み込みシステムにおいて，アプリケーションの追加や削除，システムのバージョン・アップなどを組み込み側でアドレスを動的に解決することにより実行可能なミドルウェア．
- 32ビットRISCマイコンSuperHファミリのμITRON仕様OSに対応．
- ライブラリは，表示プログラムやUSB入出力プログラムなど，各アプリケーションで共有するソフトウェアで，ライブラリの追加や削除が行える．
- システムに組み込まれているライブラリを参照することができ，ダウンロードしたプログラムからも利用可能．

●価格：¥850,000（評価版）
¥1,200,000（量産版）



■（株）ルネサスソリューションズ
TEL：03-5201-5022

●インストーラ開発統合パッケージ

Wise for Windows Installer 5J

- マウス・クリックを中心とした，設定項目が目的別に整理されたインストール・エキスパート環境を提供.
- VB5.0/6.0，VB.NET，C#，J#のインポートや既存の.MSIファイルから新たにインストール・プロジェクトの生成が可能.
- シーケンスとアクションを制御するMSIスクリプトを使用し，外部プログラムとの連携や条件分岐をサポート.
- インストーラの外観や表示動作の確認は，実際にインストールすることなく行える.
- プロフェッショナル版では，付属のデバッガにより，実行されるアクションの1ステップごとの確認，プロパティ値の変化の監視が可能．インターネット対応のインストーラの作成や定期的にWebサイトをチェックし，パッチを適用する機能を組み込むことが可能.

●価格：
スタンダード版　：¥108,000/¥128,000（ダウンロード/パッケージ）
プロフェッショナル版：¥208,000/¥228,000（ダウンロード/パッケージ）

■グレープシティ（株）
TEL：022-777-8211 FAX：022-777-8233
E-mail：sales@grapecity.com

●データ・ハンドリング・ミドルウェア

Information Wharf

- デバイスやアプリケーションなどのデータ・ソース層と業務アプリケーションや各種サービスなどのアプリケーション層を連携するためのミドルウェア.
- 企業データの一元管理，ビジネス・プロセスの自動化を分散環境において実現.
- マルチデバイス・データに対応し，一定の形式にしたがって管理が可能.
- 収集したデータの状況に応じて，自動的なビジネス・プロセスやサービスの起動が可能.
- EPCグローバルやユビキタスIDセンタが策定しているサービスやソフトウェア・コンポーネントを呼び出し，標準化技術に対応することが可能.
- 業務アプリケーションの共通基盤とすることで，シンプルな企業システムの構築が可能.

●価格：¥9,000,000～

■日本ユニシス（株）
E-mail：newsrelease-box@unisys.co.jp

●デバイス・ドライバ自動生成ツール

MakeApp for H8Tiny

- マイクロコントローラ周辺モジュールをサポートする，ビジュアル開発ツール.
- 使用，不使用のポート・ピンの最新状態をカラーコードで示し，グラフィカルなピン使用表示をサポート.
- ポイント・アンド・クリック動作だけで，周辺モジュールのデバイス・ドライバ（初期化，ランタイム，割り込みハンドラ）のCソース・コードを自動生成.
- Project Explorerでデバイス・ドライバ関数や割り込みハンドラを含め，プロジェクト状態の概要を表示.
- 最適コード・ジェネレータが，効率的でテストされたソース・コードを生成.
- ルール・チェッカと特殊機能レジスタ値の自動計算で，開発期間とデバッグ作業を削減可能.
- システムに必要なデバイス・ドライバ関数だけを使用することによりスペースを節約.
- HTMLフォーマットによるプロジェクト・レポートの自動生成.

●価格：下記へ問い合わせ

■（株）プロトン ソフトボート事業部
TEL：03-5337-6430 FAX：03-5337-6130

●開発ツール

インテル C++ソフトウェア開発ツールスイート

- Intel XScaleテクノロジに基づくアプリケーションを構築する開発者をターゲットとしており，Intel PXA25x/26xアプリケーション・プロセッサ，および次世代のプロセッサのサポートが含まれる.
- Metroworks CodeWarrior，ワークベンチ，C++コンパイラ，アセンブラ，リンカ，デバッガのIDEサポートなどの機能をサポート.
- XDBデバッガには，シミュレータ，JTAG，ROMモニタ・デバッガが含まれる.
- RTOS XDBプラグインには，Palm OS，Symbian OS，Nucleus OSのサポートが含まれる.
- ICEでユーザのターゲットやコードをデバッグできるインターフェース・ソフトウェア「WATCHPOINT for Intel SDT」を開発．これによって，開発者は洗練されたデバッグ・オプションを得て，UniSTAC IIフルICE/JTAG ICEおよび次世代機EJ-Debugの機能を使用することが可能.

●価格：¥398,000

■（株）ソフィアシステムズ
TEL：044-989-7245 FAX：044-989-7005

●Linux開発プラットホーム

MontaVista Linux Consumer Electronics Edition 3.1

- Eclipseテクノロジをベースとした，新たな統合環境であるMontaVista DevRocket1.0を組み込んでいる.
- Linuxカーネル2.4.20に対応.
- テキサスインスツルメンツ，インテル，ルネサス，モトローラなどの広範囲にわたるコンシューマ機器に特化したプロセッサをターゲットとしている.
- 複数のオープンソース・アプリケーション，ソフトウェア・コンポーネントに加え，多くの主要な商用アプリケーションとミドルウェアに対応.
- DPM，XIP機能，フラッシュ・メモリからのダイレクトなLinuxカーネルとアプリケーションの高速起動といった特徴を備える.
- ダイナミックな電力管理，豊富なネットワーク機能，コンシューマ機器で要求される信頼性をサポートすることでLinuxを強化.
- 低消費電力，メモリ・サイズ制御，起動/再起動時間の短縮が要求される，制約の厳しいモバイル機器の条件にも対応するようにチューニングされている.

●価格：下記へお問い合わせ

■モンタビスタソフトウェアジャパン（株）
TEL：03-5469-8840 FAX：03-5469-8801

●ネットワーク・ドライバ

GR-WinNET

- ARP，RARP，IPなどの低レベル・パケット送受信をサポート.
- 組み込み向けネットワーク・プロトコル・スタックを使用したWindows上のアプリケーションで，上位のネットワーク・プロトコル開発が可能.
- 各種ネットワーク・プロトコル・スタックに対応.
- 複数のシミュレーション・アプリケーションとの通信が可能.
- Windows上のネットワーク・アプリケーションとの通信が可能.
- Windowsのルーティング機能を使用し，外部機器との通信が可能.
- プロトタイピングや教育用途に，Windows 2000およびWindows XPで動作.
- サンプル・ドライバを標準で添付.

●価格：下記へ問い合わせ

■（株）グレープシステム
TEL：045-222-3761 FAX：045-222-3759
E-mail：info@gr.grape.co.jp

■弊誌では新製品に関するニュース・リリースを募集しております.
宛先は，〒170-8461 東京都豊島区巣鴨1-14-2 Interface編集部ニュース・リリース係
FAX：(03)5395-2127，E-mail：mngnews@cqpub.co.jp　（編集部）

読者の広場

Interfaceへの声



2004年4月号特集「作りながら学ぶEthernet活用技法」に関して

▷自社製組み込みRISCマイコン・システムにEthernetを実装し，開発しようと計画しています．通信パケットをどうやって検証しようかと考えていたところなので，本誌の特集は参考になりました．論理層コントローラ内蔵のRISCマイコンを採用しますが，近い将来には論理層をFPGAで開発しようと夢見ております!!（白石 隆）

▷Ethernetの信号レベルでの解説ははじめての経験だったので，非常にうれしい特集でした．自作LANカードも基本を知るうえで役に立ちました．FPGAの将来的な活用のヒントになりそうです．（moto）

［編］これまで本誌でのEthernet関連の特集は，下層でもEthernetパケット・レベルでしたが，この特集ではもっと物理的なレベルから理解するために，信号線レベルまで降りて解説してみました．いかがだったでしょうか．

▷今，まさにどこのネットワーク・コントローラを採用しようか検討しているところです．FPGAではなく，CPUのローカル・バスに接続できるタイプの市販のコントローラの使い方を解説して欲しいと思います．（NICセレクタ）

［編］実際の製品開発においては，市販のコントローラを使うことが多いと思われます．別の機会に，代表的なネットワーク・コントローラのローカル・バスへの接続方法や制御レジスタの詳細，ドライバの書きかたなど，実際の使いかたの詳細を解説したいと考えています．ご期待ください．

▷普段は市販のネットワーク・コントローラを使っているので，実際のLANケーブルを流れている信号がどのようなものかは気にしていませんでした．プリアンブルやFCSなどはコントローラが自動で処理してくれますし，衝突検出や再送処理も特殊な場合を除いて，一般的にはソフトウェアで処理する必要はありません．ネットワーク・コントローラの中身がどのように動いているかを理解することができました．（組み込みドライバ屋）

アンケートの結果

興味のあった記事（2004年4月号で実施）

①第1章 Ethernetの基礎知識
②第2章 10Base-Tの詳細
③プロローグ Ethernetのハードウェアを理解しよう！
④やり直しのための信号数学（第22回）
⑤第3章 10Base-T対応Ethernetカードの設計/製作
⑥第5章 Linuxネットワーク・ドライバの作成法
⑦第6章 組み込みTCP/IPプロトコル・スタックTINET詳解
⑧ハッカーの常識的見聞録（第40回）
⑨シニアエンジニアの技術草子（参拾七之段）
⑩プログラミングの要（第10回）
⑪Appendix 1 Power over Ethernetの概要
⑫第4章 10Base-T対応Ethernetカードの動作確認試験
⑬開発技術者のためのアセンブラ入門（第24回）

特集担当デスクから

☆最近になって二足歩行ロボットの発表が相次ぎました．メーカも研究機関も来年に控えた「愛・地球博」（愛知万博）でアピールするためにも，今年，認知度を高めようとしているように見えます．

☆独立行政法人 新エネルギー・産業技術総合開発機構（通称NEDO技術開発機構）が「平成16年度　21世紀ロボットチャレンジプログラム　次世代ロボット実用化プロジェクトに係る委託・助成先の公募」（平成16年2月25日）を実施しています．「21世紀においては，社会の少子高齢化の進展やサービス経済化の流れに沿って，消費者の生活分野，公共分野，医療福祉分野からさまざまな産業分野に至るまでの幅広い活動を支援するロボットへの期待が高まっている」とし，「これら生活分野の中で，当面，大きな市場ニーズを有するものは，コミュニケーション，警備，掃除などを行うロボットであると考えられるが，これらの実現に当たっては，個々の要素技術の開発に加え，現実的な使用環境において十分に機能する実用化システム化に向けての開発研究と，それを評価し，さらなる高度化につなげるための技術実証が不可欠である」との見方を示し，「また，さらに中長期的な視点からは，産業界や大学などにおいて取り組まれている新たなロボット技術に係る実用的なアイデアを発掘し，一般市民のロボットへの理解を深めながら当該技術をプロトタイプとして開発し，幅広いロボット関連産業の振興とロボット技術の発展を図っていくことが必要である」としています．

☆プロトタイプ開発支援事業には1件あたり最大3000万円×50件が用意されているとのことです．5月中旬には採用委託先の発表があるので，本誌でもご紹介できると思います．

☆今回は各ロボットを出しているメーカ・研究機関にロボットに使われている詳細な技術内容についてのアンケートを実施しましたが，「残念ながら技術内容については，まだ公表できません」というところもありました．しかし，ロボットを構成するための技術はすべて組み込み技術に含まれるものです．今後も今回取り上げた通信技術やOSの技術，制御技術などを随時取り上げてゆきます．



⑭第7章 TCP/IPプロトコル・スタックの開発と性能評価
⑮SDIOカード開発入門(第5回)
⑯TMS320C6713搭載DSPスターターキットを使ったC++によるDSPオブジェクト指向プログラミング(第3回)
⑰IPパケットの隙間から(第61回)
⑱「VxWORKS」を使ったRTOS技術の基礎と応用(第5回)
⑲第21回エレクトロテスト・ジャパン
⑳Engineering Life in Silicon Valley

特集『作りながら学ぶEthernet活用技法』についてのアンケートの結果

Q1 Ethernetに対応した機器の設計開発をしたことがありますか?
①はい(10%)
②いいえ(80%)

(Q1で「はい」と回答された方に質問です)
Q2 ネットワーク・インターフェース・コントローラ(NIC)には,どのような形態のデバイスを使いましたか?
①単体NIC(50%)
②NIC内蔵CPU(50%)
③FPGA + PHYデバイス(0%)
④その他(0%)

Q3 Ethernetに関連した内容で,どんな記事を希望しますか?(複数回答可)
①規格解説(12%)

②ギガビットの技術動向(15%)
③代表的NICの制御方法解説(15%)
④Ethernet搭載マイコン・ボードの開発事例(9%)
⑤プロトコルスタック開発/移植事例(9%)
⑥各種プロトコル解説(20%)
⑦TCP/IPアプリケーション・プログラミング(15%)
⑧サーバ・アプリケーション・プログラミング(5%)
⑨その他(0%)

Interface

次号予告

2004年7月号は 5月25日発売です

MIPS/R2000/R3000/PMC-RMシリーズ/Alchemyシリーズ/V_Rシリーズ/TXシリーズ

『MIPSプロセッサ徹底活用技法』

地上ディジタル放送が開始され，各種セットトップ・ボックス/ディジタル・チューナ，そしてHDD/DVDビデオ・レコーダなど，従来の家電とは比較にならないほど高機能な情報家電機器が次々と登場している．PCと連携したり，ネットワークに接続できるなど，これら情報家電に要求される処理能力は，これまでの組み込み向けプロセッサでは対応できないほどの高い性能が要求される．このように，これまでとは次元の違う処理性能を要求されたとき，どんなプロセッサを採用すべきだろうか．

MIPSプロセッサは，そのような用途に最適なプロセッサである．200MHzや400MHzといったクロック周波数では，ほかのアーキテクチャのプロセッサよりも消費電力が低く，また，下は数十MHzから，上は500MHzや1GHzという周波数まで，MIPSアーキテクチャのプロセッサがさまざまなメーカから供給されている．

次回の特集では，MIPSの歴史や情報家電機器への採用状況などの現状から，基本的なMIPSアーキテクチャの解説，そして各社のMIPSアーキテクチャ・プロセッサについて解説する．

編集後記

●ロボットと聞いてなにを思い浮かべますか．日本人の場合は間違いなく鉄腕アトムのような人型ロボットだそうです．自動車工場にある溶接ロボットを見ると，日本の産業技術の高さを誇る光景が思い浮かばれますが，アピール度はやはり二足歩行の人型ロボットには負けます．来年の愛知万博が楽しみです．（檀）

●最近，送信元メール・アドレスを詐称したウィルス付きspamが激増した感じがします．先日も某社広報の方から「あなたのPC，ウィルスに感染してませんか？」というメールを頂いたり，さらにはウィルス・メールを受信すると送信元に「感染してませんか」メールを自動送信するところもあるようで…それはやり過ぎでは？（M）

●あまりにも突然に引っ越しが決まってしまい，慌ただしい毎日を送っている．作業量自体は大したこともないのだが，残された時間のことを考えると，それだけで嫌になってくる．行政サービスに電話，インターネット，ケーブルTVなど，全部登録申請をしなければならない．この十年で持って行く物が増えた．（＝I0）

●「美しいプログラム」を目指すのは当たり前だと思っていたが，最近の若い人はこの概念が理解できないと友人のプログラマが嘆いていた．プログラムはたんなる飯の種であり，そこに美意識などは入る余地がないのだとか．バグが減り保守性が上がるという利点を説いたそうだが，若手プログラマは理解してくれたのだろうか．（み）

●私も日本人の例に漏れず「ロボット」と聞いて真っ先に二足歩行ロボット(「魁!!クロマティ高校」に出てくる「メカ沢くん」)を思い浮かべた．その次に思い浮かべたものは自動販売機．これもロボットなんだそうな．前に科学館の人が「自動販売機はロボットですが，ガンダムは違います」って言ってました．（もみ）

●キムタク主演の「プライド」が終わりました．優秀なアイスホッケー選手の話でしたが，さすがドラマ．女性ファンがいっぱいいて，派手な応援もあって，人気スポーツで（笑）．実際はチーム数が少なくなり国内だけでは存続が危ぶまれているので，このドラマ（キムタク）効果で少しでもファンが増えると良いのですが．（Y2）

●1歳半になる女の子供がいる．最近うつぶせになると発作がとまらない症状が頻繁に起こるので，総合病院に行って診てもらうことにした．親として可能性が高い原因を何とかして医者から引き出そうとするが，煮え切らない答えばかり．「結果責任」を考える前に「説明責任」を全うしてほしいものだ．（ちゃん）

●自宅のテレビの調子が悪く，買い換えを検討しています．しかしいざとなると，薄型の大画面がいいだの，どうせならばホームシアターにしたいだの，さまざまな欲望が出てきてしまい，結局は「もう少し安くなるまで待ってみよう」で購入に至らず．幸いまだ映るし，もうしばらくは夢を見させてもらおうかな．（と）

お知らせ

■ 読者の広場

本誌に関するご意見・ご希望などを，綴じ込みのハガキでお寄せください．読者の広場への掲載分には粗品を進呈いたします．なお，掲載に際しては表現の一部を変更させていただくことがありますので，あらかじめご了承ください．

■ 投稿歓迎

本誌に投稿をご希望の方は，連絡先（自宅/勤務先）を明記のうえ，テーマ，内容の概要をレポート用紙1～2枚にまとめて「Interface投稿係」までご送付ください．メールでお送りいただいても結構です（送り先はsupportinter@cqpub.co.jpまで）．追って採否をお知らせいたします．なお，採用分には小社規定の原稿料をお支払いいたします．

■ 本誌掲載記事についてのご注意

本誌掲載記事には著作権があり，示されている技術には工業所有権が確立されている場合があります．したがって，個人で利用される場合以外は，所有者の許諾が必要です．また，掲載された回路，技術，プログラムなどを利用して生じたトラブルについては，小社ならびに著作権者は責任を負いかねますので，ご了承ください．

本誌掲載記事をCQ出版（株）の承諾なしに，書籍，雑誌，Webといった媒体の形態を問わず，転載，複写することを禁じます．

■ コピー・サービスのご案内

本誌バックナンバーの掲載記事については，在庫（原則として24か月分）のないものに限りコピー・サービスを行っています．コピー体裁は雑誌見開きの，複写機による白黒コピーです．なお，コピーの発送には多少時間がかかる場合があります．

●コピー料金（税込み）
1ページにつき100円

●発送手数料（判型に関わらず）
1～10ページ：100円，11～30ページ：200円，31～50ページ：300円，51～100ページ：400円，101ページ以上：600円

●送付金額の算出方法
総ページ数×100円＋発送手数料

●入金方法
現金書留か郵便小為替による郵送

●明記事項
雑誌名，年月号，記事タイトル，開始ページ，総ページ数

●宛て先
〒170-8461　東京都豊島区巣鴨1-14-2
CQ出版株式会社　コピーサービス係
（TEL：03-5395-4211，FAX：03-5395-1642）

■ お問い合わせ先のご案内

●在庫，バックナンバー，年間購読送付先変更に関して
販売部：03-5395-2141

●広告に関して
広告部：03-5395-2133

●雑誌本文に関して
編集部：03-5395-2122

記事内容に関するご質問は，返信用封筒を同封して編集部宛てに郵送してくださるようお願いいたします．筆者に回送してお答えいたします．

Interface

2004年6月号　第30巻　第6号　通巻第324号）
2004年6月1日発行（毎月1日発行）
定価は裏表紙に表示してあります

発行人／増田久喜
編集人／山形孝雄
編集／大野典宏　村上真紀　山口光樹　大竹友美　小林由美子
デザイン・DTP／クニメディア株式会社
表紙デザイン／株式会社プランニング・ロケッツ
本文イラスト／神崎真理子　森　祐子
広告／澤辺　彰　中元正夫　菅原利江

発行所／CQ出版株式会社　〒170-8461　東京都豊島区巣鴨1-14-2
電話／編集部（03）5395-2122　FAX／（03）5395-2127
広告部（03）5395-2133　URL　http://www.cqpub.co.jp/interface/
販売部（03）5395-2141　E-mail　supportinter@cqpub.co.jp
CQ Publishing Co.,Ltd.／1-14-2 Sugamo, Toshima-ku, Tokyo 170-8461, Japan
印刷／クニメディア株式会社　美和印刷株式会社
製本／星野製本株式会社


日本ABC協会加盟誌
（新聞雑誌部数公査機構）　ISSN0387-9569

本誌は有名書店ほか，コンビニエンス・ストア「セブン-イレブン」でも予約購読ができます．詳しくは，お店にお問い合わせください．